

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

April 2008

Beta Draft

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

E10224-01

Copyright © 2005, 2008, Oracle. All rights reserved.

Primary Author: Virginia Beecher, Deanna Bradshaw, Rima Dave, Mark Kennedy, and Alex Prazma

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

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

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

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

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

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

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

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

Alpha and Beta Draft documentation are considered to be in prerelease status. This documentation is intended for demonstration and preliminary use only. We expect that you may encounter some errors, ranging from typographical errors to data inaccuracies. This documentation is subject to change without notice, and it may not be specific to the hardware on which you are using the software. Please be advised that prerelease documentation is not warranted in any manner, for any purpose, and we will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

Contents

Preface	xxvii
Audience	xxvii
Documentation Accessibility	xxvii
Related Documents	xxviii
Conventions	xxviii

Part I SOA Suite Introduction

1 Service-Oriented Architecture and Oracle SOA Suite

1.1	Introduction to Service-Oriented Architecture.....	1-1
1.2	Introduction to Oracle SOA Suite.....	1-1
1.3	Introduction to Oracle SOA Suite Components	1-3
1.3.1	Oracle Enterprise Service Bus	1-4
1.3.1.1	Oracle Mediator	1-4
1.3.1.2	Service Infrastructure.....	1-5
1.3.1.3	Oracle Adapters	1-5
1.3.1.4	Business Events and the Events Delivery Network	1-6
1.3.1.5	Oracle Business Rules	1-6
1.3.1.6	Oracle Policy Manager.....	1-6
1.3.1.7	Metadata Service Repository	1-6
1.3.2	Oracle BPEL Process Manager.....	1-6
1.3.2.1	About Using Oracle ESB with Oracle BPEL Process Manager	1-7
1.3.3	Human Task	1-7
1.3.4	Oracle Business Activity Monitoring.....	1-7
1.3.5	Oracle Complex Event Processing	1-7
1.3.6	Oracle User Messaging Service.....	1-8
1.3.6.1	Messaging Processing Engine.....	1-8
1.3.6.2	User Messaging Preferences	1-8
1.3.6.3	Messaging API	1-8
1.3.6.4	Messaging Drivers.....	1-8
1.3.6.5	Sample Applications	1-9
1.3.7	Separately Licensed Products	1-9
1.3.7.1	Oracle JDeveloper.....	1-9
1.3.7.2	Universal Description Discovery and Integration.....	1-9
1.3.7.3	Oracle Business Process Analysis Suite.....	1-10

1.3.7.4	Oracle Data Integrator	1-10
1.3.7.5	Oracle Business Intelligence.....	1-10
1.3.7.6	Oracle B2B.....	1-10
1.4	Introduction to an SOA Composite Application Using SCA Technologies.....	1-11
1.4.1	Binding Components	1-14
1.4.2	Service Engines and Service Components	1-14
1.4.3	Deployed Service Archives	1-15
1.5	How to Use This Guide.....	1-15

2 Introduction to the SOA Composite Editor

2.1	Introduction to the SOA Composite Editor	2-1
2.1.1	Introduction to Services	2-3
2.1.2	Introduction to a Composite and the SCA Descriptor	2-4
2.1.3	Introduction to Service Components	2-6
2.1.4	Introduction to References	2-8
2.1.5	Introduction to Wires	2-9
2.1.6	Introduction to SOA Composite Application in Oracle JDeveloper	2-9
2.2	Creating an SOA Project in Oracle JDeveloper	2-10
2.2.1	Task 1: Creating an Application	2-11
2.2.2	Task 2: Creating an SOA Composite.....	2-11
2.2.3	Task 3: Adding a Service Component	2-14
2.2.3.1	What You May Need to Know About Adding and Deleting a Service Component.. 2-16	
2.2.4	Task 4: Editing a Service Component	2-16
2.2.5	Task 5: Adding a Service	2-18
2.2.5.1	What You May Need to Know About Adding and Deleting Services	2-21
2.2.6	Task 6: Wiring a Service and a Service Component	2-21
2.2.6.1	What You May Need to Know About Adding and Deleting Wires	2-22
2.2.7	Task 7: Adding a Reference.....	2-22
2.2.7.1	What You May Need to Know About Adding and Deleting References	2-23
2.2.8	Task 8: Wiring a Service Component and a Reference.....	2-24
2.2.9	Task 9: Updating Message Schemas of Components (Optional).....	2-25
2.2.9.1	What You May Need to Know About Updating Message Schemas of Components 2-25	
2.2.10	Task 10: Deploying the Project.....	2-26
2.2.11	Task 11: Testing the SOA Composite Application.....	2-26

3 Life Cycle of an Application

3.1	Introduction to Application Life Cycles	3-1
3.2	Methods for Designing Applications	3-1
3.3	Considerations for Collaborative Development.....	3-2
3.4	Deploying Applications	3-2
3.4.1	Deploying Applications with Oracle JDeveloper	3-2
3.4.1.1	Task 1: Configuring an SOA Project Deployment Profile (Optional)	3-3
3.4.1.2	Task 2: Creating an Application Deployment Profile (Optional)	3-3
3.4.1.3	Task 3: Deploying the Application Profile.....	3-5
3.5	Testing Applications.....	3-8

3.6	Troubleshooting Applications	3-8
3.6.1	Recovering from Faults.....	3-8
3.6.1.1	Design Time Configuration.....	3-8
3.6.1.2	BPEL Process and Oracle Mediator Fault Handling Features	3-10
3.6.1.3	Recovering from Faults in Oracle Enterprise Manager Grid Control Console	3-11
3.6.2	Viewing Log Files	3-11
3.6.2.1	Log File and Location.....	3-11
3.6.2.2	Log Level Settings	3-12
3.6.2.3	Setting Logging Levels.....	3-13
3.6.2.4	Writing Logs to Text Files	3-15
3.6.2.5	Displaying Debug Information	3-15
3.6.3	Viewing Deployment Error Files.....	3-16

Part II Enterprise Service Bus Infrastructure

4 XSLT Mapper and Transformations

4.1	Use Case for Transformation.....	4-1
4.2	Creating an XSL Map File	4-1
4.2.1	Creating a New XSL Map File in Oracle BPEL Process Manager.....	4-2
4.2.2	Creating an XSL Map File from Imported Source and Target Schema Files in Oracle BPEL Process Manager	4-3
4.2.3	Creating an XSL Map File in Oracle Mediator	4-5
4.3	Introduction to the XSLT Mapper	4-8
4.3.1	XSLT Creation Overview.....	4-9
4.3.2	Notes on the XSLT Mapper	4-12
4.4	Using the XSLT Mapper.....	4-12
4.4.1	Adding Additional Sources.....	4-13
4.4.2	Simple Copy by Linking Nodes	4-14
4.4.3	Setting Constant Values	4-14
4.4.4	Adding Functions	4-15
4.4.4.1	Editing Function Parameters	4-16
4.4.4.2	Chaining Functions	4-16
4.4.4.3	Named Templates	4-17
4.4.4.4	Importing User-Defined Functions.....	4-17
4.4.5	Editing XPath Expressions	4-19
4.4.6	Adding XSLT Constructs.....	4-21
4.4.6.1	Conditional Processing with xsl:if	4-21
4.4.6.2	Conditional Processing with xsl:choose.....	4-22
4.4.6.3	Creating Loops with xsl:for-each	4-23
4.4.6.4	Cloning xsl:for-each	4-24
4.4.6.5	Copying Nodes with xsl:copy-of.....	4-24
4.4.6.6	Including External Templates with xsl:include	4-25
4.4.7	Automatically Mapping Nodes	4-25
4.4.7.1	Auto Map with Confirmation.....	4-27
4.4.8	Viewing Unmapped Target Nodes	4-28
4.4.9	Generating Dictionaries	4-29
4.4.10	Creating Map Parameters and Variables	4-29

4.4.10.1	Creating a Map Parameter	4-30
4.4.10.2	Creating a Map Variable.....	4-30
4.4.11	Searching Source and Target Nodes	4-31
4.4.12	Controlling Generation of Unmapped Target Elements.....	4-32
4.4.13	Ignoring Elements in the XSLT Document	4-32
4.4.14	Replacing a Schema in the XSLT Mapper	4-32
4.4.15	Using Type Substitution in the XSLT Mapper	4-33
4.5	Testing the Map.....	4-35
4.5.1	Test XSL Map Window	4-36
4.5.2	Generating Reports.....	4-38
4.5.2.1	Correcting Memory Errors When Generating Reports.....	4-39
4.5.3	Sample XML Generation	4-39
4.6	Summary	4-40

5 Getting Started with Oracle Mediator

5.1	Introduction to Oracle Mediator.....	5-1
5.2	Overview of Mediator Designer Environment.....	5-2
5.3	Creating a Mediator Component.....	5-5
5.3.1	Creating a Mediator Component Without Interface Definition.....	5-5
5.3.1.1	How to Create a Mediator With No Interface Definition	5-5
5.3.2	Creating a Mediator Based on a WSDL File	5-5
5.3.2.1	How to Create a Mediator based on a WSDL File	5-5
5.3.3	Creating a Mediator with One-Way Interface Definition.....	5-6
5.3.3.1	How to Create a Mediator with One-Way Interface Definition	5-6
5.3.3.2	What Happens When You Create a Mediator Component with One-Way Interface Definition 5-6	
5.3.4	Creating a Mediator with Synchronuous Interface Definition.....	5-7
5.3.4.1	How to Create a Mediator with Synchronous Interface Definition	5-7
5.3.4.2	What Happens When You Create a Mediator Component with Synchronous Interface Definition 5-8	
5.3.5	Creating a Mediator with Asynchronuous Interface Definition	5-8
5.3.5.1	How to Create a Mediator with Asynchronous Interface Definition	5-8
5.3.5.2	What Happens When You Create a Mediator Component with Asynchronous Interface Definition 5-9	
5.3.6	Creating a Mediator for Event Subscription.....	5-10
5.3.6.1	How to Create a Mediator component for Event Subscription	5-10
5.4	Defining Interface for an Empty Mediator Component.....	5-13
5.4.1	Subscribing to Events	5-14
5.4.2	Defining Services for a Mediator component.....	5-14
5.5	Generating a WSDL File.....	5-17
5.6	Specifying Operation or Event Subscription Properties	5-19
5.7	Modifying a Mediator component	5-20
5.7.1	Modifying Mediator component Operations	5-20
5.7.2	Modifying Mediator component Event Subscriptions.....	5-21
5.8	Deleting a Mediator component.....	5-22

6 Creating Routing Rules

6.1	Introduction to Routing Rules	6-1
6.2	Defining Routing Rules.....	6-2
6.2.1	Creating Dynamic Routing Rules.....	6-4
6.2.2	Creating Static Routing Rules	6-4
6.2.2.1	Specifying Target Service	6-4
6.2.2.2	Specifying Expression for Filtering Messages.....	6-7
6.2.2.3	Specifying Sequential or Parallel Execution	6-13
6.2.2.4	Using Semantics Validation	6-13
6.2.2.5	Creating Transformations	6-14
6.2.2.6	Assigning Values	6-16
6.2.2.7	Callback, Synchronous Reply, and Fault Handling	6-19
6.2.3	Callback, Synchronous Reply, and Fault Handling.....	6-19
6.3	Use Case	6-19
6.3.1	Step-By-Step Instructions for Creating the CustomerRouter Use Case.....	6-20
6.3.1.1	Task 1: Creating an Oracle JDeveloper Application and Project.....	6-20
6.3.1.2	Task 2: Creating CustomerRouter Mediator	6-22
6.3.1.3	Task 3: Creating a File Adapter Service	6-22
6.3.1.4	Task 4: Creating External References	6-25
6.3.1.5	Task 5: Specifying Routing Rules.....	6-26
6.3.1.6	Task 6: Configuring Oracle Application Server Connection.....	6-34
6.3.1.7	Task 7: Deploying CustomerRouterProject	6-35
6.3.2	Running and Monitoring the CustomerRouterProject Application.....	6-36

7 Oracle Mediator Error Handling

7.1	Introduction to Oracle Mediator Error Handling	7-1
7.1.1	Fault Policies.....	7-1
7.1.1.1	Conditions	7-2
7.1.1.2	Actions	7-4
7.1.2	Fault Bindings	7-4
7.2	Using Error Handling.....	7-5
7.3	XML Schema Files for Error Handling	7-5
7.3.1	Fault-policies.xml Schema File	7-5
7.3.2	Fault-bindings.xml Schema File	7-9

8 Business Events and the Event Delivery Network

8.1	Introduction to Business Events	8-1
8.1.1	Local and Remote Events Boundaries	8-3
8.1.2	Synchronous Subscriptions	8-3
8.2	Using Business Events in Oracle JDeveloper.....	8-4
8.2.1	Creating a Business Event	8-4
8.2.2	Subscribing to a Business Event	8-6
8.2.3	Publishing a Business Event.....	8-8
8.2.4	Integrating ADF BC Business Events with Oracle Mediator	8-9

9 Working with Domain Value Maps

9.1	Introduction to Domain Value Maps	9-1
9.1.1	Domain Value Map Features	9-1
9.1.1.1	Qualifier Support	9-2
9.1.1.2	Qualifier Order Support	9-2
9.1.1.3	One-to-Many Mapping Support	9-3
9.2	Creating Domain Value Maps	9-4
9.2.1	How to Create Domain Value Maps	9-4
9.2.2	What Happens When You Create a Domain Value Map	9-6
9.3	Editing a Domain Value Map	9-8
9.3.1	Adding Columns to a Domain Value Map	9-9
9.3.2	Adding Rows to a Domain Value Map	9-9
9.3.3	Reordering the Columns in a Domain Value Map	9-9
9.4	Using Domain Value Map Functions	9-9
9.4.1	Understanding Domain Value Map Functions	9-9
9.4.1.1	dvm:lookupValue	9-9
9.4.1.2	dvm:lookupValue1M	9-10
9.4.2	Using Domain Value Map Functions in Transformation	9-11
9.4.3	Using a Domain Value Map Functions to Create XPath Expressions	9-14
9.4.4	What Happens at Run Time	9-15
9.5	Domain Value Map Use Case	9-15
9.5.1	Step-By-Step Instructions for Creating the Use Case	9-15
9.5.1.1	Task 1: Creating an Oracle JDeveloper Application and Project	9-16
9.5.1.2	Task 2: Creating a Domain Value Map	9-16
9.5.1.3	Task 3: Creating a File Adapter Service	9-18
9.5.1.4	Task 4: Creating ProcessOrders Mediator	9-19
9.5.1.5	Task 5: Creating a File Adapter Reference	9-20
9.5.1.6	Task 6: Specifying Routing Rules	9-21
9.5.1.7	Task 7: Configuring Oracle Application Server Connection	9-26
9.5.1.8	Task 8: Deploying the Composite Application	9-26
9.5.2	Running and Monitoring the HierarchicalValue Application	9-26

10 Working with Cross References

10.1	Introduction to Cross References	10-1
10.2	Creating and Modifying Cross Reference Tables	10-4
10.2.1	Creating a Cross Reference Table	10-4
10.2.2	Adding a Column to a Cross Reference Table	10-6
10.2.3	Deleting a Column from a Cross Reference Table	10-7
10.3	Populating Cross Reference Tables	10-7
10.3.1	xref:populateXRefRow Function	10-7
10.3.1.1	Using xref:populateXRefRow Function	10-9
10.3.2	xref:populateXRefRow1M Function	10-12
10.4	Looking Up Cross Reference Tables	10-13
10.4.1	xref:lookupXRef Function	10-13
10.4.1.1	Using xref:lookupXRef Function	10-14
10.4.2	xref:lookupXRef1M Function	10-16
10.5	Deleting a Cross Reference Table Value	10-17

10.6	Schema Definition(XSD) File for Cross References.....	10-20
10.7	Cross Reference Use Case.....	10-21
10.7.1	Introduction.....	10-21
10.7.2	Prerequisites.....	10-21
10.7.3	Step-By-Step Instructions for Creating the Use Case.....	10-22
10.7.3.1	Task 1: Configuring Oracle Database and Database Adapter.....	10-22
10.7.3.2	Task 2: Creating an Oracle JDeveloper Application and Project.....	10-23
10.7.3.3	Task 3: Creating a Cross Reference.....	10-23
10.7.3.4	Task 4: Creating a Database Adapter Service.....	10-24
10.7.3.5	Task 5: Creating EBS and SBL External References.....	10-27
10.7.3.6	Task 6: Creating Logger External Reference.....	10-30
10.7.3.7	Task 7: Creating Mediator Components.....	10-31
10.7.3.8	Task 8: Specifying Routing Rules for Mediator1.....	10-32
10.7.3.9	Task 9: Specifying Routing Rules for Common Mediator.....	10-44
10.7.3.10	Task 10: Configuring Oracle Application Server Connection.....	10-56
10.7.3.11	Task 11: Deploying the Composite Application.....	10-56
10.7.4	Running and Monitoring the XrefCustApp Application.....	10-56

Part III BPEL Process Service Component

11 Getting Started with Oracle BPEL Process Manager

11.1	Starting Oracle SOA Suite Components.....	11-1
11.2	Introduction to the BPEL Designer Environment.....	11-2
11.2.1	Introduction to BPEL Process Service Component Creation and Oracle JDeveloper.....	11-2
11.2.1.1	Application Navigator.....	11-5
11.2.1.2	Design Window.....	11-5
11.2.1.3	Source Window.....	11-7
11.2.1.4	History Window.....	11-8
11.2.1.5	Component Palette.....	11-9
11.2.1.6	Property Inspector.....	11-11
11.2.1.7	Structure Window.....	11-11
11.2.1.8	Log Window.....	11-12
11.3	Introduction to Activities.....	11-12
11.4	Introduction to Partner Links.....	11-14
11.5	Partner Link Creation and the SOA Composite Editor.....	11-15
11.5.1	Creating a Partner Link For an Outbound Adapter.....	11-15
11.5.2	Creating a Partner Link for an Inbound Adapter.....	11-16
11.5.3	Creating a Partner Link from an Abstract WSDL to Call a Service.....	11-16
11.5.4	Creating a Partner Link from an Abstract WSDL to Implement a Service.....	11-16
11.5.5	Creating a Human Task or Decision Service.....	11-17
11.5.6	Creating a Partner Link From an Existing Human Task, Decision Service, or Mediator Routing Service.....	11-18
11.6	Introduction to Oracle BPEL Server.....	11-18
11.7	Introduction to Oracle BPEL Process Manager Technology Adapters.....	11-18

12 Manipulating XML Data in BPEL Processes

12.1	Use Cases for Manipulating XML Data in BPEL.....	12-1
12.2	Introduction to Manipulating XML Data in BPEL Concepts	12-2
12.2.1	How XML Data Works in BPEL	12-2
12.2.2	About Data Manipulation and XPath Standards	12-2
12.3	Delegating XML Data Operation to Data Provider Services	12-4
12.3.1	Addressing XML Data Operation Challenges.....	12-5
12.3.2	How the Entity Variable Works	12-6
12.3.2.1	Inbound Direction	12-6
12.3.2.2	Outbound Direction	12-7
12.3.3	Example of Defining an Entity Variable.....	12-7
12.3.3.1	Create an Entity Variable and Choose a Partner Link	12-7
12.3.3.2	Create a Binding Key	12-8
12.4	Initializing a Variable with Expression Constants or Literal XML.....	12-9
12.5	Copying Between Variables	12-10
12.6	Accessing Fields Within Element-Based and Message Type-Based Variables	12-10
12.7	Assigning Numeric Values.....	12-11
12.8	Mathematical Calculations with XPath Standards	12-11
12.9	Assigning String Literals.....	12-12
12.10	Concatenating Strings	12-12
12.11	Assigning Boolean Values	12-13
12.12	Assigning Date or Time	12-13
12.13	Manipulating Attributes	12-14
12.14	Manipulating XML Data with bpelex Extensions	12-14
12.14.1	bpelex:append	12-15
12.14.2	bpelex:insertBefore	12-15
12.14.3	bpelex:insertAfter	12-16
12.14.4	bpelex:remove	12-17
12.14.5	bpelex:rename and XSD Type Casting	12-18
12.14.6	bpelex:copyList	12-19
12.15	Validating XML Data with bpelex:validate	12-20
12.16	Manipulating XML Data Sequences That Resemble Arrays	12-20
12.16.1	Statically Indexing into an XML Data Sequence That Uses Arrays	12-21
12.16.2	Determining Sequence Size	12-22
12.16.3	Dynamically Indexing by Applying a Trailing XPath to an Expression	12-22
12.16.3.1	Dynamic Indexing Example.....	12-22
12.16.3.2	Using the bpelex:append Extension to Append New Items to a Sequence.....	12-23
12.16.3.3	Merging Data Sequences	12-24
12.16.3.4	Generating Functionality Equivalent to an Array of an Empty Element.....	12-24
12.16.4	Limited Support for SOAP-Encoded Arrays	12-25
12.17	Converting from a String to an XML Element.....	12-25
12.18	Differences Between Document-Style and RPC-Style WSDL Files	12-26
12.19	Manipulating SOAP Headers in BPEL	12-27
12.19.1	Receiving SOAP Headers in BPEL.....	12-27
12.19.2	Sending SOAP Headers in BPEL.....	12-28

13 Invoking a Synchronous Web Service from BPEL Processes

13.1	Use Case for Synchronous Web Services.....	13-1
13.2	Introduction to Synchronous Service Concepts	13-2
13.2.1	Establishing the Partner Link.....	13-2
13.2.1.1	Defining the Partner Link in the BPEL Code.....	13-2
13.2.1.2	Using the WSDL File to Enable the Web Services to Work with a BPEL Process Service Component	13-3
13.2.2	Using the Invoke Activity to Perform a Request	13-4
13.3	Calling a Synchronous Service.....	13-5
13.4	Summary	13-6

14 Invoking an Asynchronous Web Service from BPEL Processes

14.1	Use Case for Asynchronous Web Services.....	14-1
14.2	Introduction to Asynchronous Callback Concepts.....	14-3
14.2.1	partnerLinkTypes for Asynchronous Services.....	14-3
14.2.2	Calling the Service from BPEL.....	14-4
14.2.3	How the Invoke and Receive Activities Work	14-5
14.2.4	Managing Multiple Active BPEL Process Service Component Instances Using Correlation Methods	14-6
14.2.4.1	WS-Addressing	14-7
14.2.4.2	Using Correlation Sets to Coordinate Asynchronous Message Body Contents.....	14-9
14.2.5	Using the Reply Activity to Send Messages in Response to a Receive Activity ...	14-10
14.2.6	Using Dehydration Points to Maintain Long-Running Asynchronous Processes	14-10
14.3	Calling an Asynchronous Service.....	14-10
14.3.1	Step 1: Adding a Partner Link for an Asynchronous Service.....	14-10
14.3.2	Step 2: Adding an Invoke Activity	14-11
14.3.3	Step 3: Adding a Receive Activity	14-13
14.3.4	Step 4: Performing Additional Activities	14-13
14.4	Using Correlation Sets in an Asynchronous Service	14-14
14.4.1	Step 1: Creating a Project	14-14
14.4.2	Step 2: Configuring Partner Links and File Adapter Services	14-15
14.4.2.1	Creating an Initial Partner Link and File Adapter Service.....	14-15
14.4.2.2	Creating a Second Partner Link and File Adapter Service.....	14-17
14.4.2.3	Creating a Third Partner Link and File Adapter Service.....	14-18
14.4.3	Step 3: Creating Three Receive Activities.....	14-19
14.4.3.1	Creating an Initial Receive Activity	14-19
14.4.3.2	Creating a Second Receive Activity	14-20
14.4.3.3	Creating a Third Receive Activity	14-20
14.4.4	Step 4: Creating Correlation Sets.....	14-21
14.4.4.1	Creating an Initial Correlation Set	14-21
14.4.4.2	Creating a Second Correlation Set	14-21
14.4.5	Step 5: Associating Correlation Sets with Receive Activities	14-22
14.4.5.1	Associating the First Correlation Set with a Receive Activity	14-22
14.4.5.2	Associating the Second Correlation Set with a Receive Activity	14-22
14.4.5.3	Associating the Third Correlation Set with a Receive Activity	14-23
14.4.6	Step 6: Creating Property Aliases.....	14-23

14.4.6.1	Creating Property Aliases for NameCorr	14-23
14.4.6.2	Creating Property Aliases for IDCorr.....	14-24
14.4.7	Step 7: Reviewing WSDL File Content	14-25
14.5	Summary	14-25
15	Parallel Flow in BPEL Processes	
15.1	Use Case for Parallel Flows	15-1
15.2	Introduction to Parallel Flow Concepts.....	15-1
15.3	Customizing the Number of Flow Activities by Using the flowN Activity.....	15-2
15.3.1	BPEL Code Example of the FlowN Activity	15-4
15.4	Summary	15-6
16	Conditional Branching in BPEL Processes	
16.1	Use Case for Conditional Branching.....	16-1
16.2	Introduction to Conditional Branching Concepts.....	16-1
16.3	Using a Switch Activity to Define Conditional Branching	16-2
16.4	Using a While Activity to Define Conditional Branching.....	16-4
16.5	Summary	16-4
17	Fault Handling in BPEL Processes	
17.1	Use Case for Fault Handling	17-1
17.2	Defining a Fault Handler	17-1
17.3	BPEL Standard Faults.....	17-3
17.4	Categories of BPEL Faults.....	17-3
17.4.1	Business Faults	17-3
17.4.2	Run-time Faults.....	17-3
17.4.2.1	bindingFault	17-4
17.4.2.2	remoteFault.....	17-5
17.4.2.3	replayFault.....	17-5
17.4.2.4	Catching Run-time Faults Example	17-5
17.5	Getting Fault Details with the getFaultAsString XPath Extension Function	17-5
17.6	Using the Scope Activity to Manage a Group of Activities	17-6
17.7	Throwing Internal Faults	17-6
17.8	Returning External Faults	17-7
17.8.1	Returning a Fault in a Synchronous Interaction	17-7
17.8.2	Returning a Fault in an Asynchronous Interaction	17-7
17.9	Using a Fault Handler within a Scope	17-7
17.9.1	Using the Empty Activity to Insert No-Op Instructions into a Business Process ...	17-8
17.10	Using Compensation After Undoing a Series of Operations	17-8
17.11	Using the Terminate Activity to Stop a Business Process Instance	17-9
18	Incorporating Java and J2EE Code in BPEL Processes	
18.1	Introduction to Java and J2EE Code in BPEL Concepts.....	18-1
18.1.1	Using Java Code Wrapped as a SOAP Service.....	18-1
18.1.2	Directly Embedding Java Code in a BPEL Process.....	18-2
18.1.2.1	Using the bpelx:exec Tag to Embed Java Code Snippets into a BPEL Process.	18-2

18.1.2.2	Using an XML Facade to Simplify DOM Manipulation	18-3
18.1.2.3	bpel:exec Built-in Methods.....	18-4
18.1.3	Using Java Code Wrapped in a Service Interface	18-4
18.2	Using Java Embedding in a BPEL Process	18-5
18.3	Using PL/SQL Procedures with SOA Applications.....	18-6
18.3.1	Initiating a SOA Application with PL/SQL Procedures.....	18-6
18.3.2	Accessing a Service Implemented as a PL/SQL Procedure	18-7
18.4	Summary	18-7

19 Events and Timeouts in BPEL Processes

19.1	Use Case for Events and Timeouts.....	19-1
19.2	Introduction to Event and Timeout Concepts	19-1
19.3	Using the Pick Activity to Select Between Continuing a Process or Waiting	19-2
19.4	Using the Wait Activity to Set an Expiration Time.....	19-4
19.5	Setting Timeouts for Synchronous Processes	19-4
19.6	Defining a Timeout.....	19-4
19.7	Summary	19-6

20 Invoking a BPEL Process Service Component

20.1	Use Case for Invoking a BPEL Process Service Component	20-1
20.2	Sending Messages to a BPEL Process Service Component.....	20-1
20.2.1	Invoking a BPEL Process Service Component with the Web Service/SOAP Interface	20-2

21 Coordinating Master and Detail Processes

21.1	Introduction to Master and Detail Process Coordinations	21-1
21.2	Master and Detail Process Definitions in the BPEL Files.....	21-3
21.2.1	BPEL File Definition for the Master Process	21-4
21.2.1.1	Correlating a Master Process with Multiple Detail Processes	21-5
21.2.2	BPEL File Definition for Detail Processes	21-6
21.3	Coordinating Master and Detail Processes in Oracle JDeveloper	21-6
21.3.1	Create a Master Process	21-6
21.3.2	Create a Detail Process.....	21-8
21.3.3	Create an Invoke Activity.....	21-10

22 Interaction Patterns in BPEL Processes

22.1	One-Way Message	22-1
22.2	Synchronous Interaction	22-2
22.3	Asynchronous Interaction	22-3
22.4	Asynchronous Interaction with Timeout	22-4
22.5	Asynchronous Interaction with a Notification Timer	22-4
22.6	One Request, Multiple Responses	22-5
22.7	One Request, One of Two Possible Responses	22-6
22.8	One Request, a Mandatory Response, and an Optional Response.....	22-7
22.9	Partial Processing.....	22-8

22.10	Multiple Application Interactions	22-9
22.11	Summary	22-10

23 Notifications and the Oracle User Messaging Service

23.1	Use Cases for Notifications.....	23-1
23.2	Introduction to Oracle User Messaging Service and Notification Concepts.....	23-1
23.2.1	Reliable Notifications	23-3
23.3	Configuring the Service in Oracle JDeveloper.....	23-3
23.3.1	The E-mail Notification Channel.....	23-5
23.3.1.1	Setting E-mail Attachments	23-6
23.3.1.2	Sending Actionable E-mails	23-8
23.3.1.3	Formatting the Body of an E-mail Message as HTML	23-9
23.3.2	The SMS Notification Channel	23-9
23.3.3	The Voice Notification Channel.....	23-10
23.3.4	Setting E-mail Addresses and Telephone Numbers Dynamically	23-11
23.3.5	Selecting Notification Recipients by Browsing the User Directory	23-12
23.3.6	Setting Automatic Replies to Unprocessed Messages.....	23-12
23.3.7	XML Validation Failure with the Oracle User Messaging Service	23-13
23.4	Summary	23-13

24 Oracle BPEL Process Manager Sensors

24.1	Use Cases for Sensors	24-1
24.2	Introduction to Sensor Concepts	24-1
24.3	Implementing Sensors and Sensor Actions in Oracle JDeveloper.....	24-2
24.3.1	Configuring Sensors	24-3
24.3.2	Configuring Sensor Actions	24-6
24.3.3	Publishing to Remote Topics and Queues	24-8
24.3.4	Creating a Custom Data Publisher.....	24-8
24.3.5	Registering the Sensors and Sensor Actions in composite.xml	24-10
24.4	Sensors and Oracle Enterprise Manager 11g Application Server Control Console	24-11
24.4.1	Viewing Sensor and Sensor Action Definitions	24-11
24.4.2	Viewing Sensor Data	24-14
24.5	Sensor Integration with Oracle Business Activity Monitoring	24-15
24.5.1	Creating a Connection to Oracle BAM Server.....	24-16
24.5.2	Creating a Sensor	24-17
24.5.3	Creating a BAM Sensor Action.....	24-17
24.6	Sensor Public Views.....	24-19
24.6.1	BPM Schema	24-20
24.7	Sensor Actions XSD File.....	24-23
24.8	Summary	24-32

25 Business Rule Service Component

25.1	Business Rule Concepts.....	25-1
25.1.1	Business Rules and Business Rule Engines.....	25-1
25.1.2	Business Rule Service Component	25-2
25.1.3	Oracle SOA Suite	25-2

25.2	Business Rule Architecture.....	25-3
25.2.1	Business Rule Service Component Metadata File.....	25-4
25.2.2	Rule Dictionary	25-5
25.2.3	SCA Component Type	25-6
25.2.4	Decision Services.....	25-6
25.2.4.1	Decision Services that Expose an RL Function.....	25-9
25.2.5	Stateful Interactions with a Decision Component	25-12
25.3	Use Cases for Integration of Business Processes and Business Rules	25-13
25.4	Integration of BPEL Processes, Human Tasks, and Business Rules	25-13
25.4.1	Business Rule Service Component	25-14
25.4.2	Business Rule Activity of a Business Process	25-16
25.4.3	Human Task Component	25-18
25.5	Deploying a Business Rule	25-19
25.6	Running a Business Rule in a SOA Composite Application	25-20

Part IV Human Workflow Service Component

26 Designing Human Tasks

26.1	Introduction to Workflow Services	26-1
26.1.1	Workflow Functionality: A Procurement Process Example.....	26-4
26.2	Use Cases for Workflow Services	26-5
26.2.1	Assigning a Task to a User or Role	26-5
26.2.2	Using the Various Participant Types	26-6
26.2.3	Escalation, Expiration, and Delegation	26-6
26.2.4	Automatic Assignment and Delegation	26-7
26.2.5	Work Queues and Proxy Support	26-7
26.2.6	The Oracle BPEL Worklist Application.....	26-7
26.3	Workflow Services Components	26-8
26.4	Participant Types in Workflow Services	26-10
26.4.1	Chaining Multiple Tasks.....	26-11
26.5	Introduction to the Modeling Process.....	26-11
26.5.1	Create a Human Task Definition with the Human Task Editor.....	26-12
26.5.2	Optionally Associate the Human Task Definition with a BPEL Process.....	26-12
26.5.3	Generate the Task Display Form	26-12
26.6	Creating the Human Task Definition with the Human Task Editor.....	26-12
26.6.1	Creating a Human Task Service Component and Accessing the Human Task Editor	26-13
26.6.2	Reviewing the Sections of the Human Task Editor	26-15
26.6.3	Specifying the Task Title, Priority, Outcome, and Owner	26-16
26.6.3.1	Specifying a Task Title and Priority.....	26-16
26.6.3.2	Specifying a Task Outcome.....	26-17
26.6.3.3	Specifying a Task Description	26-18
26.6.3.4	Specifying a Task Category.....	26-18
26.6.3.5	Specifying a Task Owner.....	26-18
26.6.4	Specifying the Task Payload Data Structure	26-21
26.6.5	Assigning Task Participants.....	26-22
26.6.5.1	Specifying Task Approvers.....	26-23

26.6.5.2	Configuring the Single Approver Participant Type	26-24
26.6.5.3	Configuring the Group Vote Participant Type	26-27
26.6.5.4	Configuring the Management Chain Participant Type	26-31
26.6.5.5	Configuring the Sequential List of Approvers Participant Type.....	26-34
26.6.5.6	Configuring the FYI Assignee Participant Type	26-37
26.6.5.7	Configuring the External Routing Service Participant Type.....	26-38
26.6.5.8	Allowing All Participants to Invite Other Participants.....	26-40
26.6.5.9	Adding Reviewers	26-40
26.6.5.10	Routing Tasks to All Participants in the Specified Order.....	26-40
26.6.5.11	Abruptly Completing a Condition.....	26-41
26.6.5.12	Advanced Task Routing Using Business Rules	26-42
26.6.6	Escalating, Renewing, or Ending the Task.....	26-46
26.6.6.1	Introduction to Escalation and Expiration Policy	26-47
26.6.6.2	Never Expire Policy.....	26-49
26.6.6.3	Expire After Policy	26-49
26.6.6.4	Renew After Policy.....	26-50
26.6.6.5	Escalate After Policy.....	26-51
26.6.6.6	Specifying a Due Date.....	26-52
26.6.7	Specifying Participant Notification Preferences.....	26-53
26.6.7.1	Notifying Recipients of Changes to Task Status	26-54
26.6.7.2	Editing the Notification Message	26-55
26.6.7.3	Setting Up Reminders	26-55
26.6.7.4	Securing Notifications, Making Messages Actionable, and Sending Attachments	26-56
26.6.8	Specifying Advanced Settings	26-57
26.6.8.1	Specifying Escalation Rules.....	26-58
26.6.8.2	Specifying WordML Style Sheets for Attachments	26-59
26.6.8.3	Specifying Style Sheets for Attachments.....	26-59
26.6.8.4	Specifying Multilingual Settings	26-59
26.6.8.5	Overriding Default Exception Management	26-60
26.6.8.6	Specifying Callback Classes on Task Status	26-61
26.6.8.7	Allowing Task and Routing Customization in BPEL Callbacks.....	26-62
26.6.8.8	Specifying a Workflow Signature Policy	26-62
26.6.8.9	Specifying Access Rules on Task Content.....	26-63
26.6.9	Specifying Annotations.....	26-68
26.6.10	Exiting the Human Task Editor and Saving Your Changes	26-69
26.7	Associating the Human Task Service Component with a BPEL Process	26-69
26.7.1	Associating a Human Task with a BPEL Process.....	26-69
26.7.2	Defining the Human Task Activity Title, Initiator, Priority, and Parameter Variables	26-70
26.7.2.1	Specifying the Task Title.....	26-71
26.7.2.2	Specifying the Task Initiator and Task Priority	26-72
26.7.2.3	Specifying Task Parameters	26-72
26.7.3	Viewing the Generated Human Task Activity	26-73
26.7.3.1	BPEL Callbacks	26-75
26.7.4	Defining the Human Task Activity Advanced Features	26-77
26.7.4.1	Specifying a Scope Name and a Global Task Variable Name.....	26-78
26.7.4.2	Specifying a Task Owner.....	26-78

26.7.4.3	Specifying an Identification Key	26-78
26.7.4.4	Including the Task History of Other Human Tasks	26-78
26.7.4.5	Using Task and Routing Customizations in BPEL Callbacks	26-79
26.7.5	Outcome-Based Modeling	26-80
26.7.5.1	Payload Updates	26-80
26.7.5.2	Case Statements for Other Task Conclusions	26-80
26.8	End-to-End Workflow Examples	26-81
26.8.1	Help Desk Request Example	26-82
26.8.2	Prerequisites	26-82
26.8.3	Modeling the Help Desk Request	26-82
26.8.3.1	Creating an Application	26-83
26.8.3.2	Creating the SOA Project	26-83
26.8.3.3	Create the Human Task Service Component	26-85
26.8.3.4	Designing the Human Task	26-86
26.8.3.5	Associating the Human Task and BPEL Process Service Components	26-88
26.8.3.6	Deploying the SOA Composite Application	26-90
26.8.3.7	Initiating the Process Instance	26-90
26.8.3.8	Creating an Oracle BPM Worklist SOA Project	26-90
26.8.3.9	Designing the Task Display Form	26-92
26.8.3.10	Resolving the Task in Oracle BPM Worklist	26-98

27 Designing Task Display Forms for Human Tasks

27.1	Introduction to the Task Display Form	27-1
27.2	Creating an ADF Task Flow Based on a Human Task Definition	27-2
27.2.1	How to Create an ADF Task Flow Within the Same Application as the Human Task..... 27-2	
27.2.2	How to Create an ADF Task Flow Within the Same Composite Application as the Human Task 27-3	
27.2.3	What Happens When You Create an ADF Task Flow Based on a Human Task Definition 27-4	
27.3	Creating a Task Display Form	27-5
27.3.1	Custom Drop Handlers	27-5
27.3.1.1	Task Header	27-6
27.3.1.2	Task History	27-7
27.3.1.3	System Actions	27-7
27.3.1.4	Custom Actions	27-8
27.3.1.5	Task Comments	27-9
27.3.1.6	Task Attachments	27-9
27.3.1.7	Complete Task without Payload	27-9
27.3.1.8	Complete Task with Payload	27-9
27.3.1.9	Task Display for Notification	27-10
27.3.2	Standard Drop Handlers	27-10
27.3.3	How to Create a Task Display Form Using Individual Drop Handlers	27-11
27.3.4	How to Create a Task Display Form Using the Complete Task with Payload Drop Handler 27-17	
27.3.5	What Happens When You Create a Task Display Form	27-18
27.4	Creating an E-Mail Notification	27-18

27.4.1	How to Create an E-Mail Notification.....	27-19
27.4.1.1	Creating a Task Flow with a Router	27-19
27.4.1.2	Creating an E-Mail Notification Page.....	27-22
27.4.2	What Happens When You Create an E-Mail Notification Page	27-24
27.5	Deploying a Composite Application with a Task Flow	27-24
27.5.1	Before Deploying the Task Display Form: Port Changes	27-24
27.5.2	How to Deploy a Composite Application with a Task Flow	27-26
27.5.3	How to Deploy a Task Flow as a Separate Application.....	27-27
27.5.4	What Happens When You Deploy the Task Display Form.....	27-28
27.6	Displaying a Task Display Form in the Worklist	27-28
27.6.1	How to Display the Task Display Form in the Worklist.....	27-29
27.7	Troubleshooting the Task Display Form.....	27-30

28 Human Task Services

28.1	Human Task Services	28-1
28.1.1	EJB, SOAP, and Java Support for the Human Task Services.....	28-1
28.1.2	Security Model for Services.....	28-3
28.1.2.1	Limitation on Propagating Identity to Workflow Services when Using SOAP Web Services 28-3	
28.1.2.2	Security in EJBs	28-3
28.1.2.3	Creating Human Task Context on Behalf of a User	28-4
28.1.3	Task Service	28-4
28.1.4	Task Query Service.....	28-7
28.1.5	Identity Service.....	28-8
28.1.5.1	Identity Service Providers	28-10
28.1.6	Notification Service	28-11
28.1.7	Task Metadata Service	28-11
28.1.8	User Metadata Service.....	28-12
28.1.9	Runtime Config Service	28-13
28.1.9.1	Internationalization of Attribute Labels.....	28-15
28.1.10	Digital Signatures and the Evidence Store Service	28-16
28.1.10.1	Prerequisites	28-17
28.1.10.2	Interfaces and Methods	28-18
28.2	Notifications from Human Workflow	28-20
28.2.1	Configuring the Notification Channel.....	28-21
28.2.2	Contents of Notification.....	28-22
28.2.3	Configuring Notification Messages in Different Languages.....	28-23
28.2.4	Sending Actionable Messages.....	28-24
28.2.4.1	Sending Actionable E-mails for Human Tasks	28-24
28.2.4.2	Sending Actionable Instant Messages	28-26
28.2.5	Error Message Support	28-26
28.2.6	Sending Inbound and Outbound Attachments.....	28-27
28.2.7	Sending Inbound Comments	28-27
28.2.8	Reliability Support.....	28-27
28.2.9	Sending Secure Notifications	28-28
28.2.10	Channels Used for Notifications	28-28
28.2.11	Notification Services for SMS, Voice Mail, and IM Channels	28-28

28.2.12	Sending Reminders.....	28-28
28.2.13	Custom Notification Headers	28-29
28.2.14	Managing the Notification Service.....	28-29
28.3	Configuring the Assignment Service	28-30
28.3.1	Dynamic Assignment Functions	28-30
28.3.1.1	Implementing a Dynamic Assignment Function.....	28-31
28.3.1.2	Configuring Dynamic Assignment Functions	28-31
28.3.1.3	Configuring Display Names for Dynamic Assignment Functions	28-32
28.3.2	Dynamically Assigning Task Participants with the Assignment Service	28-32
28.3.2.1	Assignment Service Overview	28-33
28.3.2.2	Implementing an Assignment Service.....	28-33
28.3.2.3	Example of Assignment Service Implementation.....	28-34
28.3.2.4	Deploying a Custom Assignment Service	28-36
28.3.3	Custom Escalation Function.....	28-36
28.4	Human Task Service and Identity Service Related XPath Extension Functions	28-36
28.4.1	Deprecated Human Task Service and Identity Service Functions	28-37
28.5	NLS Configuration.....	28-38
28.6	Changes to APIs	28-38
28.7	Summary	28-38

29 Using Oracle BPM Worklist

29.1	Introduction to Oracle BPM Worklist	29-1
29.2	Logging In to Oracle BPM Worklist	29-2
29.2.1	How to Log In to the Worklist	29-2
29.2.2	What Happens When You Log In to the Worklist.....	29-3
29.3	Customizing the Task List Page	29-5
29.3.1	How to Filter Tasks.....	29-5
29.3.2	How to Create and Customize Worklist Views	29-7
29.3.3	How to Customize the Task Status Chart	29-12
29.3.4	How to Create a To-Do Task.....	29-13
29.4	Acting on Tasks: The Task Details Page	29-15
29.4.1	System Actions.....	29-17
29.4.2	Task History	29-18
29.4.3	How to Act on Tasks	29-20
29.4.4	How to Act on Tasks That Require a Digital Signature	29-24
29.5	Setting Vacation and Other Rules.....	29-28
29.5.1	How to Set Vacation and Other Rules	29-28
29.6	Using the Worklist Administration Functions	29-33
29.6.1	How to Manage Other Users' or Groups' Rules (as an Administrator)	29-33
29.6.2	How to Set the Worklist Display (Application Preferences).....	29-35
29.6.3	How to Map Flex Fields.....	29-36
29.7	Creating Worklist Reports	29-40
29.7.1	How to Create Reports.....	29-41
29.7.2	What Happens When You Create Reports	29-42
29.7.2.1	Unattended Tasks Report.....	29-43
29.7.2.2	Tasks Priority Report	29-43
29.7.2.3	Tasks Cycle Time Report	29-44

29.7.2.4	Tasks Productivity Report.....	29-45
29.8	Accessing Oracle BPM Worklist in Local Languages.....	29-45
29.8.1	How to Change the Language Used in the Worklist.....	29-46
29.8.2	How to Change the Time Zone Used in the Worklist.....	29-47
29.9	Summary	29-47

30 Human Task and Approval Management Integration

n	Introduction to AMX	30-1
n	AMX Components.....	30-3
n	Human Task and AMX Concepts.....	30-4
n	Approval Task.....	30-4
n	Approver Groups	30-4
n	Approver List	30-4
n	Chain	30-4
n	Configuration Variables	30-4
n	Dimensions (or Message Attribute Group)	30-7
Figure 30–2	Human Task Editor	30-7
Figure 30–2	List Builders.....	30-7
n	List Builder Usage	30-8
n	Notification.....	30-8
n	Parts of an Approver List	30-8
Figure 30–3	Routing Slip	30-9
Figure 30–3	Stages.....	30-9
Figure 30–4	.task file	30-9
n	User Task	30-10
n	Worklist Application.....	30-10
n	Decision Service and Oracle Business Rules Concepts	30-10
n	Approval Routing Policies	30-10
n	Approval Policy Type	30-10
n	Action	30-11
n	Conditions	30-11
n	Criteria.....	30-11
n	Priority	30-11
n	Response Type	30-11
n	Rule Context.....	30-11
n	Terms.....	30-12
n	Validity Period.....	30-12
n	Designing AMX Approval Tasks in Oracle JDeveloper.....	30-12
n	Review Prerequisites.....	30-13
Table 30–4	Select the Composite Approver Participant Type	30-13
n	Create a Dimension	30-15
7.	Create the Stages.....	30-16
14.	Configure Approval List Builders for a Stage	30-19
n	Arranging the Order of Approval List Builders	30-19
2.	Deleting Approval List Builders	30-20
2.	Adding Approval List Builders.....	30-21
3.	Adding Approval List Builder Policy Actions.....	30-21

31 Human Task and Microsoft Excel Integration

31.1	Invoking a BPEL Process from Excel Workbook.....	31-1
31.2	Attaching Excel Workbooks to E-mail Notifications	31-2

Part V Oracle Business Activity Monitoring

32 Creating Data Objects

32.1	Defining Data Objects.....	32-1
32.1.1	Adding Fields.....	32-2
32.1.2	Adding Lookup Fields	32-3
32.1.3	Adding Calculated Fields.....	32-4
32.1.4	Adding Time Stamp Fields.....	32-4
32.2	Adding Permissions on Data Objects	32-4
32.2.1	Copying Permissions from Other Data Objects	32-5
32.3	Viewing Existing Data Objects.....	32-6
32.3.1	Viewing Data Object General Information	32-6
32.3.2	Viewing Data Object Layouts	32-7
32.3.3	Viewing Data Object Contents.....	32-7
32.4	Using Data Object Folders	32-8
32.4.1	Creating Folders.....	32-8
32.4.2	Working with Folders	32-8
32.4.3	Setting Folder Permissions	32-9
32.4.4	Moving Folders	32-10
32.4.5	Renaming Folders.....	32-10
32.4.6	Deleting Folders.....	32-10
32.5	Adding Security Filters	32-10
32.5.1	Copying Security Filters from Other Data Objects	32-12
32.6	Adding Dimensions.....	32-12
32.6.1	Time Dimensions	32-13
32.7	Renaming and Moving Data Objects	32-14
32.8	Adding Indexes	32-14
32.9	Clearing Data Objects.....	32-15
32.10	Deleting Data Objects	32-15
32.11	System Data Objects	32-15

33 Using External Data Sources

33.1	Introducing External Data Sources.....	33-1
33.2	Listing External Data Sources	33-1
33.3	Defining External Data Sources	33-2
33.4	Editing External Data Sources.....	33-2
33.5	Deleting External Data Sources	33-2
33.6	External Data Source Example.....	33-2

34 Creating Alerts

34.1	Introducing Alerts.....	34-1
------	-------------------------	------

34.2	Creating Alert Rules	34-1
34.3	Using Alert Rule Options	34-2
34.3.1	Events	34-2
34.3.2	Conditions.....	34-3
34.3.3	Actions.....	34-3
34.3.4	Frequency Constraint.....	34-4
34.4	Creating Alert Rules From Templates	34-5
34.5	Creating Alert Rules With Messages	34-5
34.6	Creating Complex Alerts	34-6
34.7	Modifying Rules for Alerts.....	34-6
34.8	Viewing Alert History.....	34-7
34.9	Clearing Alert History.....	34-7
34.10	Activating Alerts.....	34-7
34.11	Launching Alerts by URL.....	34-8
34.12	Deleting Alerts.....	34-8
34.13	Parameterized Alerts.....	34-8

35 Using the Oracle BAM Data Control

35.1	Introduction to the Oracle BAM Data Control	35-1
35.2	Creating Oracle BAM Server Connections.....	35-1
35.2.1	How to Modify Oracle BAM Data Control Connections to Oracle BAM Servers ..	35-2
35.3	Creating Projects That Can Use Oracle BAM Data Controls.....	35-3
35.4	Exposing Oracle BAM with Oracle ADF Data Controls	35-4
35.4.1	How to Create Oracle BAM Data Controls.....	35-4
35.4.2	What Happens in Your Project When You Create an Oracle BAM Data Control ..	35-5
35.4.2.1	How an Oracle BAM Data Control Appears in the Data Controls Panel	35-5
35.5	Creating Oracle BAM Data Control Queries	35-6
35.5.1	Choosing a Query Type	35-6
35.5.2	How to Create Parameters	35-7
35.5.3	How to Create Calculated Fields.....	35-7
35.5.3.1	Creating Groups in Calculated Fields	35-8
35.5.4	How to Select, Organize, and Sort Fields.....	35-8
35.5.5	How to Create Filters	35-8
35.5.5.1	How to Create Filter Headers.....	35-8
35.5.5.2	How to Create Filter Entries	35-9
35.5.5.3	Entering Comparison Values.....	35-10
35.5.5.4	Using Active Now	35-11
35.5.6	How to Select and Organize Groups	35-11
35.5.6.1	How to Configure Time Groups and Time Series	35-12
35.5.7	How to Create Aggregates	35-13
35.5.8	How to Modify the Query	35-13
35.6	Using Oracle BAM Data Controls in ADF Pages	35-13

36 Creating Enterprise Message Sources

36.1	Introducing Enterprise Message Sources	36-1
36.2	Listing Enterprise Message Sources.....	36-2
36.3	Defining Enterprise Message Sources.....	36-2

36.3.1	Using Advanced XML Formatting.....	36-4
36.3.2	XSL Processing and Example Code	36-5
36.3.2.1	XSL Processing in an Example Enterprise Message Source.....	36-5
36.3.2.2	Handling Complex Messages	36-6
36.3.2.3	Phoenix Debt Order Example	36-7
36.3.2.4	Sequence Numbers.....	36-8
36.3.2.5	XSLT Code	36-8
36.4	Editing Enterprise Message Sources	36-10
36.5	Copying Enterprise Message Sources.....	36-10
36.6	Deleting Enterprise Message Sources	36-10

37 Using ICommand

37.1	Introducing ICommand	37-1
37.2	Executing ICommand.....	37-1
37.3	General Command and Option Syntax	37-2
37.3.1	Specifying the Command	37-2
37.4	Object Name Syntax	37-2
37.5	Command-line-only Parameters.....	37-3
37.6	Running ICommand Remotely	37-4
37.7	Summary of Individual Commands	37-5
37.8	Detailed Command Descriptions	37-6
37.8.1	Export	37-6
37.8.2	Import.....	37-10
37.8.3	Delete.....	37-13
37.8.4	Rename.....	37-14
37.8.5	Clear.....	37-15
37.9	Format of Command File.....	37-15
37.9.1	Inline Content.....	37-16
37.9.2	Command IDs	37-16
37.9.3	Continue On Error	37-17
37.10	Format of Log File.....	37-17
37.11	Sample Export File.....	37-18
37.12	Regular Expressions	37-19
37.13	Using ICommand Web Service.....	37-21
37.13.1	Differences between the ICommand Web Service and the ICommand Command-Line Utility	37-22
37.13.2	Using the ICommand Web Service	37-22
37.13.3	Security Issues	37-23
37.13.3.1	IIS Security (HTTP 401 error) [remove for 11g?]	37-23
37.13.3.2	Active Data Cache Security.....	37-23

Part VI Oracle User Messaging Service

38 Configuring User Messaging Service

38.1	Overview of Oracle User Messaging Service Configuration.....	38-1
38.2	Configuring User Messaging Service Drivers	38-2

38.2.1	Configuring Driver Properties.....	38-4
38.2.2	Configuring the E-Mail Driver	38-6
38.2.3	Configuring the SMPP Driver.....	38-10
38.2.4	Configuring the XMPP Driver	38-12
38.2.5	Configuring the VoiceGenie Driver.....	38-15
38.2.6	Configuring the Proxy Driver.....	38-16
38.3	Deploying Drivers	38-17
38.4	Configuring Messaging Preferences	38-20
38.4.1	Managing Devices	38-21
38.4.1.1	Creating a Device.....	38-21
38.4.1.2	Editing a Device.....	38-22
38.4.1.3	Deleting a Device.....	38-22
38.4.1.4	Setting a Default Device	38-22
38.4.2	Creating Contact Rules using Filters	38-22
38.4.2.1	Creating Filters.....	38-23
38.4.2.2	Editing a Filter.....	38-25
38.4.2.3	Deleting a Filter.....	38-25

39 Parlay X Web Services Multimedia Messaging API

39.1	Overview of Parlay X Messaging Operations.....	39-1
39.2	Send Message Interface.....	39-1
39.2.1	sendMessage Operation.....	39-2
39.2.2	getMessageDeliveryStatus Operation	39-3
39.3	Receive Message Interface	39-3
39.3.1	getReceivedMessages Operation.....	39-4
39.3.2	getMessage Operation.....	39-5
39.3.3	getMessageURIs Operation.....	39-5
39.4	Oracle Extension to Parlay X Messaging.....	39-6
39.4.1	ReceiveMessageManager Interface	39-6
39.4.1.1	startReceiveMessage Operation	39-6
39.4.1.2	stopReceiveMessage Operation.....	39-7
39.5	Parlay X Messaging Client API and Client Proxy Packages.....	39-7

40 User Messaging Preferences

40.1	Introduction	40-1
40.1.1	Terminology	40-1
40.1.2	Configuration of Notification Delivery Preferences.....	40-2
40.1.3	Delivery Preference Rules	40-2
40.1.3.1	Data Types.....	40-2
Table 40–1	System Terms	40-3
40.1.3.2	Business Terms.....	40-3
40.1.4	Rule Actions.....	40-4
40.2	Configuring Filters and Conditions	40-4
40.2.1	Sample Conditions.....	40-4
40.2.2	Rule Activation Flow.....	40-4
40.3	Configuring Device Destination Addresses.....	40-5

41 Send Message to User Specified Channel Sample Application

41.1	Overview	41-1
41.1.1	Provided Files.....	41-1
41.2	Installing and Configuring SOA and User Messaging Service	41-2
41.3	Building the Sample	41-2
41.4	Creating a Deployment Profile	41-15
41.5	Creating a New Application Server Connection.....	41-16
41.6	Deploying the Application	41-18
41.7	Configuring Drivers	41-21
41.8	Configuring User Messaging Preferences	41-22
41.9	Testing the Sample.....	41-23
41.9.1	Verifying the Execution of Sending the E-mail.....	41-24

42 Oracle User Messaging Service Expense Report Use Case

42.1	Overview of the Expense Report Use Case.....	42-1
42.1.1	Prerequisites	42-2
42.1.1.1	Required Files.....	42-2
42.1.1.2	Configuring the User Messaging Service E-Mail Driver to Send and Receive Messages 42-5	
42.1.1.3	Creating the Database Tables that Define the Business Components	42-6
42.2	Deploying the Pre-Built Sample Applications.....	42-10
42.2.1	Deploying the SOA Composite and Task Flow Applications (ExpenseReportCompositeApp) 42-10	
42.2.2	Deploying the ADF BC Application (ExpenseReportADFBCApp)	42-11
42.3	Running the Sample Application	42-11
42.4	Building the Expense Report Applications	42-13
42.4.1	Creating the ADF BC Application (ExpenseReportADFBCApp).....	42-13
42.4.1.1	Creating the Business Component Project.....	42-13
42.4.1.2	Exposing the ExpenseReport Object as a Web Service	42-18
42.4.1.3	Creating the Deployment Profile for the ExpenseReportModel Project	42-20
42.4.1.4	Creating the ADF View Project	42-22
42.4.1.5	Creating the Deployment Profile for the ADF BC Application	42-26
42.4.2	Creating the SOA Composite Application (ExpenseReportComposite)	42-27
42.4.2.1	Creating the Mediator.....	42-27
42.4.2.2	Creating the BPEL Process	42-30
42.4.2.3	Wiring the Mediator to the BPEL Process.....	42-32
42.4.2.4	Transforming the Data.....	42-34
42.4.2.5	Creating the Human Workflow.....	42-36
42.4.2.6	Deploying the Composite Application.....	42-40
42.4.3	Creating the Task Flow Application (ExpenseAppHumanTaskFlow)	42-40
42.4.3.1	Creating the Task Flow Project.....	42-41

Part VII Composite Test Framework

43 Testing SOA Composite Applications

43.1	Overview of the Composite Test Framework.....	43-1
------	---	------

43.1.1	Test Cases Overview	43-2
43.1.2	Test Suites Overview	43-2
43.1.3	Emulations Overview	43-2
43.1.4	Assertions Overview	43-2
43.1.5	BPEL Process Code Coverage Overview	43-3
43.2	Components of a Test Suite	43-3
43.2.1	Process Initiation.....	43-3
43.2.2	Emulations	43-4
43.2.3	Assertions.....	43-5
43.2.4	Message Files	43-5
43.3	Creating Test Suites and Test Cases in Oracle JDeveloper	43-6
43.4	Creating the Contents of Test Cases.....	43-8
43.4.1	Initiating Inbound Messages.....	43-9
43.4.2	Emulating Outbound Messages	43-10
43.4.3	Emulating Callback Messages	43-13
43.4.4	Emulating Fault Messages.....	43-15
43.4.5	Creating Value or XML Assertions	43-16
43.4.5.1	Variable Assertions	43-17
43.4.5.2	XML Assertions	43-19
43.5	Deploying a Test Suite.....	43-21
43.5.1	Deploying from Oracle JDeveloper.....	43-21
43.6	Running a Test Suite and Viewing Report Results	43-22
43.6.1	Running from Oracle Enterprise Manager Fusion Middleware Control	43-22

Part VIII Appendices

A Building a Custom Worklist Client

A.1	Introduction to Building Clients for Workflow Services	A-1
A.2	Packages and Classes for Building Clients.....	A-3
A.3	Workflow Service Clients	A-3
A.3.1	The IWorkflowServiceClient Interface	A-6
A.4	Classpaths for Clients Using SOAP.....	A-6
A.5	Classpaths for Clients Using Remote EJBs.....	A-7
A.6	Classpaths for Clients Using Local EJBs.....	A-7
A.7	EJB References in Web Applications.....	A-7
A.8	Initiating a Task.....	A-8
A.8.1	Creating a Task.....	A-8
A.8.2	Creating a Payload Element in a Task.....	A-9
A.8.3	Initiating a Task Programmatically.....	A-9
A.9	Writing a Worklist Application Using the HelpDeskUI Sample	A-10

Index

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 using Oracle SOA Suite.

Documentation Accessibility

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

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

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

Related Documents

For more information, see the following Oracle resources:

- *Oracle BPEL Process Manager Administrator's Guide*

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

SOA Suite Introduction

This part introduces the components that comprise Oracle SOA Suite and how these components interact in a SOA composite application. This section also introduces the SOA Composite Editor that you use to design SOA composite applications. An introduction into the life cycle of a SOA composite application is also provided.

This part contains the following chapters:

- [Chapter 1, "Service-Oriented Architecture and Oracle SOA Suite"](#)
- [Chapter 2, "Introduction to the SOA Composite Editor"](#)
- [Chapter 3, "Life Cycle of an Application"](#)

Service-Oriented Architecture and Oracle SOA Suite

This chapter introduces Service-Oriented Architecture (SOA) and describes how Oracle SOA Suite provides support for SOA. An overview of how Oracle SOA Suite components interact in an SOA composite application is provided. An introduction to the contents of this guide is also provided.

This chapter contains the following topics:

- [Section 1.1, "Introduction to Service-Oriented Architecture"](#)
- [Section 1.2, "Introduction to Oracle SOA Suite"](#)
- [Section 1.3, "Introduction to Oracle SOA Suite Components"](#)
- [Section 1.4, "Introduction to an SOA Composite Application Using SCA Technologies"](#)
- [Section 1.5, "How to Use This Guide"](#)

See Also:

- [Chapter 2, "Introduction to the SOA Composite Editor"](#) for details on using the SOA Composite Editor to create an SOA composite application

1.1 Introduction to Service-Oriented Architecture

Changing markets, increasing competitive pressures and evolving customer needs are placing greater pressure on IT to deliver greater flexibility and speed. Today every organization is faced with the need to predict change in a global business environment, to rapidly respond to competitors, and to best exploit organizational assets for growth. In response to these challenges, leading companies are adopting SOA as a means of delivering 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.

1.2 Introduction to Oracle SOA Suite

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
- Complex event processing

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) .
- 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 (J2EE) to access business logic distributed among heterogeneous systems.
- Web Service 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.

See Also:

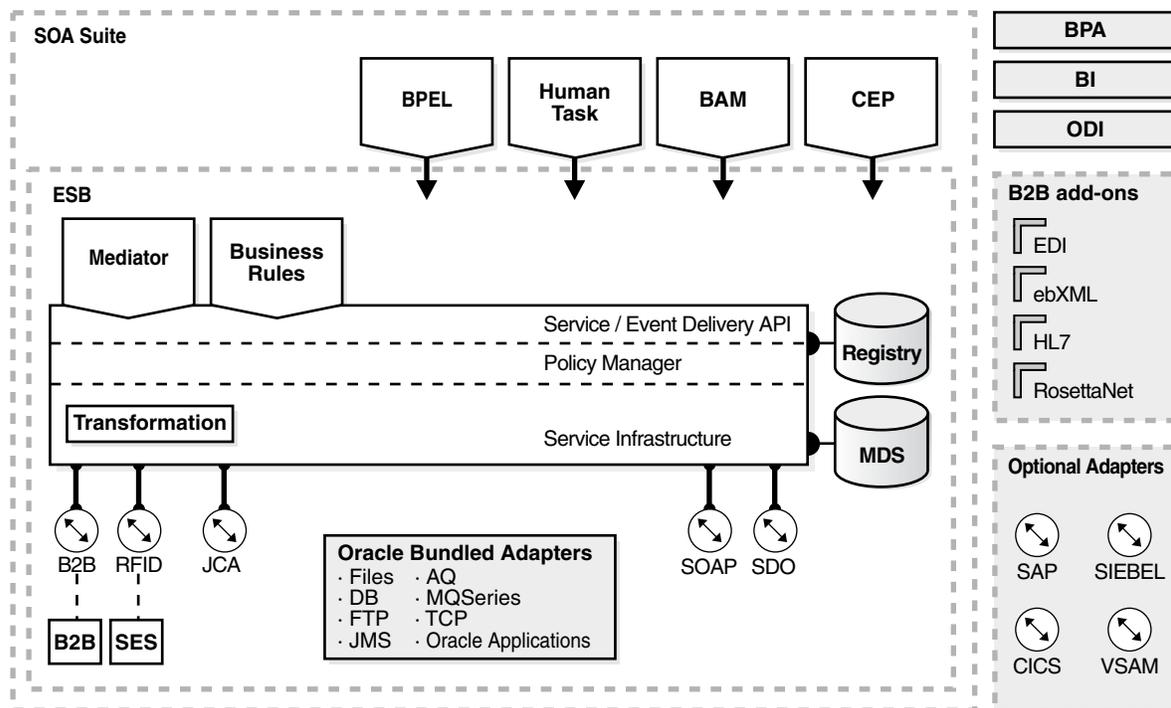
- [Section 1.4, "Introduction to an SOA Composite Application Using SCA Technologies"](#) on page 1-11
- [Chapter 2, "Introduction to the SOA Composite Editor"](#) for additional details about these key building blocks
- The following URL for SCA and SDO specifications and related material:

<http://www.osoa.org>

1.3 Introduction to Oracle SOA Suite Components

Figure 1–1 shows Oracle SOA Suite and its interoperability with other Oracle products.

Figure 1–1 Oracle SOA Suite Components



Oracle Mediator (Mediator), Oracle Business Rules (Business Rules), and Oracle Adapters plug into the Service Infrastructure, a normalized transport infrastructure, making up the Enterprise Service Bus (ESB). With the addition of the Oracle BPEL Process Manager (BPEL) and Human Task service components, the suite forms a complete Business Process Management (BPM) platform. Oracle Business Activity Monitoring (BAM) and Complex Event Processing (CEP) consume data transported over the Service Infrastructure, providing powerful business insight capabilities.

Separately licensable products, such as Oracle Business Process Analysis Suite (BPA), Oracle Business Intelligence (BI), Oracle Data Integrator (ODI), and other adapters interoperate with Oracle SOA Suite components.

The following components comprise Oracle SOA Suite:

- [Oracle Enterprise Service Bus](#)
- [Oracle BPEL Process Manager](#)
- [Human Task](#)
- [Oracle Business Activity Monitoring](#)
- [Oracle Complex Event Processing](#)
- [Oracle User Messaging Service](#)
- [Separately Licensed Products](#)

1.3.1 Oracle Enterprise Service Bus

Oracle Enterprise Service Bus (Oracle ESB) provides the following components for moving data among multiple endpoints, both within and outside of an enterprise. It uses open standards to connect, transform, and route business documents as Extensible Markup Language (XML) messages among disparate applications.

- [Oracle Mediator](#)
- [Service Infrastructure](#)
- [Oracle Adapters](#)
- [Business Events and the Events Delivery Network](#)
- [Oracle Business Rules](#)
- [Oracle Policy Manager](#)
- [Metadata Service Repository](#)

1.3.1.1 Oracle Mediator

Analogous to a load balancer routing HTTP traffic, Oracle Mediator routes incoming data to consumers. In addition, it can subscribe to and publish business events.

Using Oracle Mediator, you create routing services and rules for them. A routing service is the key component for moving a message across the enterprise service bus – from its entry point to its exit point. The rules determine how a message instance processed by the routing service gets to its next destination. Using the rules, Oracle Mediator performs the following:

- **Routing:** Determines the service engine container (BPEL process, human task, business rule, or mediator routing service) to which to send the messages
- **Validation:** Provides support for validating the incoming message payload by using a schematron or an XSD file.
- **Filtering:** If specified in the rules, applies a filter expression that specifies the contents (payload) of a message be analyzed before any service is invoked
- **Transformation:** If specified in the rules, transforms document data from one XML schema to another, thus enabling data interchange among applications using different schemas

During deployment, Oracle Mediator evaluates routing rules, performs transformations, and either invokes another service or raises another business event. A mediator routing service can handle returned responses, callbacks, faults, and time-outs. Oracle Mediator can also be used to implement a variety of integration

patterns, such as service virtualization, service aggregation, publish and subscribe, fan-in, and fan-out.

1.3.1.2 Service Infrastructure

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

- Receives messages from the service providers or external services through SOAP services or adapters, and transforms them into XML messages
- Sends messages to the appropriate service engine
- Receives messages back from the service engine and sends them to any additional service engines in the composite based on the wiring

Together, Oracle Mediator and the Service Infrastructure provide the following capabilities:

- Routing services to provide data movement
- Common data model
- Routing rules to specify routing, document transformation, and filtering
- Subscriptions to business events
- Error handling provides insight into data-movement issues

1.3.1.3 Oracle Adapters

Oracle Adapters, which are used to connect external systems to Oracle SOA Suite, follow the Java Connector Architecture (JCA) standard.

Oracle SOA Suite automatically includes the following adapters to integrate with transport protocols, data stores, and messaging middleware:

- FTP
- Java Messaging Service (JMS)
- Advanced Queuing (AQ)
- Files
- Message Queuing (MQ) Series
- Database
- Oracle Applications

Oracle provides the following separately licensed packaged-application adapters for integrating Oracle SOA Suite with various packaged applications, such as SAP and Siebel:

- Oracle Applications
- PeopleSoft
- SAP R/3
- Siebel
- J.D. Edwards OneWorld

Oracle provides the following separately licensed legacy adapters for integrating Oracle SOA Suite with legacy and mainframe applications:

- Tuxedo

- CICS
- VSAM
- IMS/TM
- IMS/DB

Note: Additional JCA adapters are provided by Oracle or other third-party companies. Visit the Oracle Technology Network for details about these adapters.

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

1.3.1.4 Business Events and the Events Delivery Network

You can raise business events when a situation of interest occurs. Business events are messages sent as the result of an occurrence or situation, such as a new order or completion of an order. In Oracle SOA Suite, Oracle Mediator is the principal tool used for subscribing to or publishing events. CEP can also subscribe to or publish events.

Business events are deployed to the Metadata Service Repository, and then published in the Event Delivery Network (EDN).

See Also: [Chapter 8, "Business Events and the Event Delivery Network"](#)

1.3.1.5 Oracle Business Rules

Oracle Business Rules enable dynamic decisions at runtime allowing, among other features, applications to rapidly adapt to regulatory and competitive pressures. This increased agility is possible because business analysts using Oracle Business Rules can create and change business rules that are separated from the application code. By using Oracle Business Rules, business analysts can change business rules without stopping business processes. Also, externalizing business rules allows business analysts to manage business rules directly, without involving programmers.

1.3.1.6 Oracle Policy Manager

Oracle Policy Manager provides security in a policy-oriented fashion. It provides the infrastructure for enforcing global security and auditing policies in the Service Infrastructure. By securing various endpoints and setting and propagating identity, it secures applications. Oracle Policy Manager provides a standard mechanism for signing messages, for doing encryption, for doing authentication, and for enabling role-based access control. Since you can declaratively change a policy without having to modify the endpoints, applying or modifying security and monitoring is a much simpler task.

1.3.1.7 Metadata Service Repository

The Metadata Service Repository stores business event definitions, business rulesets, XSLT transformations, XSD schemas, WSDL interfaces, and CEP metadata files.

1.3.2 Oracle BPEL Process Manager

BPEL provides the standard for assembling a set of discrete services into an end-to-end process flow, radically reducing the cost and complexity of process integration initiatives. Oracle BPEL Process Manager enables you to develop synchronous and

asynchronous services into end-to-end BPEL process flows. It provides process orchestration and storage of long running, asynchronous processes.

You integrate BPEL processes with external services (known as partner links). You also integrate technology adapters and services, such as human tasks, transformations, notifications, sensors, and business rules within the process.

See Also: [Part III, "BPEL Process Service Component"](#)

1.3.2.1 About Using Oracle ESB with Oracle BPEL Process Manager

SOA architects have traditionally had to choose between an ESB and a BPM platform, such as Oracle BPEL Process Manager, upon starting a new SOA project. An ESB was seen as a lightweight, fast, stateless transport and connectivity tool, while BPM provided some of these features along with more logic capabilities, often at the cost of lesser performance.

With the Oracle SOA Suite, such a choice is no longer necessary. One SOA composite application can be started as an ESB project and then enhanced with orchestration logic using BPEL, all in a single set of artifacts, deployed on a single server. Architects no longer need to worry about what to run where, and what patterns will require what product when starting a new SOA project.

A typical SOA composite application uses a Mediator to create an inbound interface that could then filter and route messages to various BPEL processes. In turn, BPEL process would use other Mediators to fan out the results.

1.3.3 Human Task

The components described previously focus mostly on integrating systems. However, most applications also require human input. People have very different characteristics than computer systems, such as response time, availability, and so on. Therefore, integration of people in business processes requires special consideration and tools. The human task component enables you to interleave human functions with connectivity to systems and services within the BPEL process flow. A human task assigns a task, such as approval for an order, to a user or role and waits for a response. The users act on the task using a worklist application. This inclusion of human tasks within a business process creates a tight integration between process and task.

See Also: [Part IV, "Human Workflow Service Component"](#)

1.3.4 Oracle Business Activity Monitoring

Oracle BAM provides business executives the ability to monitor their business services and processes in the enterprise, to correlate KPIs down to the actual business process themselves, and most important, to change business processes quickly or to take corrective action if the business environment changes.

Oracle BAM is a complete solution for building real-time operational dashboards and monitoring and alerting applications over the Web. Using this technology, business users get the ability to build interactive, real-time dashboards and proactive alerts to monitor their business services and processes.

1.3.5 Oracle Complex Event Processing

Oracle Complex Event Process is a tool that enables queries to be easily written to look for patterns in event streams. Oracle Complex Event Process listens on these streams,

caches all the necessary individual, seemingly unrelated events and tries to correlate them into specific patterns.

Users write queries using Continuous Query Language (CQL).

The potential applications of Oracle Complex Event Process are numerous, from electronic trading and risk management to intrusion detection and compliance monitoring.

The data provided from complex event processing queries has many uses, including real-time Oracle BAM dashboards.

1.3.6 Oracle User Messaging Service

Oracle User Messaging Service (UMS) provides a common service that sends messages from applications to user devices and routes incoming messages from user devices to applications.

UMS includes the following components:

- [Messaging Processing Engine](#)
- [User Messaging Preferences](#)
- [Messaging API](#)
- [Messaging Drivers](#)
- [Sample Applications](#)

1.3.6.1 Messaging Processing Engine

Oracle User Messaging Service (UMS) includes a core user messaging processing engine and client hooks that plug into both BPEL and human workflows. As a result, UMS enables you compose BPEL and human workflows that support sending messages to, or receiving messages from, any server. All of these components are J2EE-compliant.

1.3.6.2 User Messaging Preferences

UMS also supports User Messaging Preferences, an application which enables users to tailor their message delivery options by registering their devices and e-mail clients to their user accounts and by creating message filters that designate the message content sent by UMS to these devices.

1.3.6.3 Messaging API

The Messaging API library enables a client application to send and receive messages through Oracle User Messaging Service.

The Messaging Java API provides a single API that interacts with UMS through either EJB or Web Service interfaces.

1.3.6.4 Messaging Drivers

Messaging drivers implement transport protocols that transfer messages through a specified communication channel or an external protocol gateway. Oracle User Messaging Service ships with an e-mail driver already deployed to support the sending and receiving of e-mail. In addition, UMS supports SMS through the SMPP (Short Message Peer-to-Peer) driver, IM and presence through the XMPP (Extensible Messaging and Presence Protocol) driver, and TTS (text-to-speech) conversion of voice messages through its Voice driver. These drivers, which you deploy to the server

through Oracle Enterprise Manager Grid Control Console, are available in the distribution at `SOA_ORACLE_HOME/archives/applications`. *[[Question: Is this location correct??]]*

1.3.6.5 Sample Applications

User Messaging Service provides samples for developing applications that send notifications from BPEL processes and human task workflows. These applications, which are available at *[[Question: file location??]]*, provide such use-case scenarios as:

- An application containing a BPEL process that looks up a user's e-mail address and creates an e-mail message with an attachment to a the user's device, one created by the user through User Messaging Preferences.
- An application that demonstrates a BPEL process that allows a message to be sent to a user through a channel specified in User Messaging Preferences. Once a user configures a device for each supported channel and sets the default device, Oracle User Messaging Service routes the message based on these device settings.
- A help desk request where a customer files a help desk ticket assigned to a help desk agent who can then resolve this request, or route it to other help desk agents.
- An expense report notification that can be either resolved by its recipients, or routed to others.

1.3.7 Separately Licensed Products

You can use the following separately licensed products with Oracle SOA Suite:

- [Oracle JDeveloper](#)
- [Universal Description Discovery and Integration](#)
- [Oracle Business Process Analysis Suite](#)
- [Oracle Data Integrator](#)
- [Oracle Business Intelligence](#)
- [Oracle B2B](#)

See Also: [Section 1.3.1.3, "Oracle Adapters"](#) on page 1-5 for details about separately licensed adapters

1.3.7.1 Oracle JDeveloper

Oracle JDeveloper is the development component of Oracle SOA Suite. It forms a comprehensive Integrated Service Environment (ISE) for creating and deploying composite applications and managing the composite.

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

1.3.7.2 Universal Description Discovery and Integration

The Universal Description Discover and Integration (UDDI) publishes the MDS services to the outside world. UDDI provides a key component of any SOA with a configurable, scalable, secure repository of Web services that can be managed, discovered and governed by Oracle Fusion Middleware. To advertise the existence of

these services to potential consumers, you use Oracle Application Server UDDI Registry. The Oracle Application Server UDDI Registry meets the core service management needs of any enterprise:

- Enables service providers to publish and advertise their offerings
- Allows service consumers to find, access, and invoke services that meet defined criteria
- Provides critical features for SOA governance

1.3.7.3 Oracle Business Process Analysis Suite

Oracle Business Process Analysis (Oracle BPA) Suite allows process owners, business analysts, and architects to perform process modeling and analysis, simulation, and publishing of process models. The Oracle BPA Suite further supports the execution and monitoring of these process models with Oracle BPEL Process Manager and Oracle BAM.

As part of the process analysis, organizational, structural, and technological weak points in business process management (BPM) processes are revealed and improvement potential is identified.

1.3.7.4 Oracle Data Integrator

Oracle Data Integrator helps you to integrate the vast amounts of information stored in disparate systems. Oracle Data Integrator streamlines the high-performance movement and transformation of data between disparate systems in batch, real-time, synchronous, and asynchronous modes, with a focus on batch processing and large amounts of data.

Together, the Oracle SOA Suite and Oracle Data Integrator allow enterprises to perform any kind of data transfer and transformations: from real-time to batch, from small data changes propagation to complete replications. It enables companies to more easily handle initiatives related to business intelligence (BI), data warehousing, master data management (MDM), Oracle BAM, application migration and consolidation, and SOA.

1.3.7.5 Oracle Business Intelligence

Oracle Business Intelligence enables you to obtain information about your business from available data. This information helps you to understand your business better. Oracle Business Intelligence helps answer key business questions by:

- Using historical data for time-based analysis and trend analysis.
- Leveraging Oracle OLAP and Oracle Data Mining database options to provide advanced analytic features.

1.3.7.6 Oracle B2B

Oracle B2B provides business-to-business (B2B) exchange of services, information, and products. If you know who you want to trade with (for example, a specific supplier), what you want to do (for example, send a purchase order), and how you want to do it (for example, send the purchase order over the Internet), then you have defined a basic B2B transaction.

Oracle B2B interoperates with Oracle BPEL Process Manager and Oracle ESB to address an enterprise's end-to-end integration needs and exploit SOA.

1.4 Introduction to an SOA Composite Application Using 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.

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

Figure 1–2 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.

Figure 1-2 Introduction to an SOA Composite Application

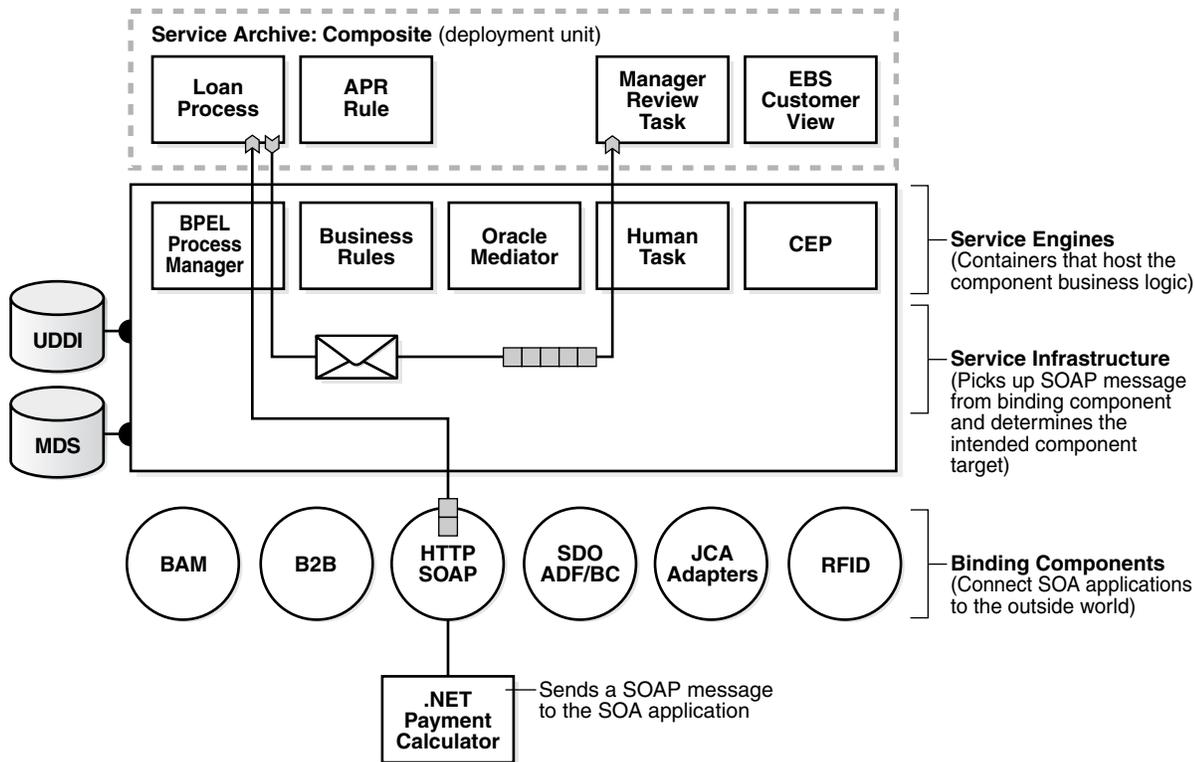


Table 1-1 describes the operability of the SOA composite application shown in Figure 1-2. References are made to sections that provide additional details.

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

Part	Description	Example of Use in Figure 1–2	See Section
Binding Components	<p>Make SOA composite applications accessible to the outside world. There are two types:</p> <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 	<p>The SOAP binding component <i>service</i>:</p> <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 1–2 is the Loan Process application. This can be an external partner link to which the BPEL service engine sends messages to obtain loan information.</p>	" Binding Components " on page 1-14
Service Infrastructure	Provides internal message transport	<p>The Service Infrastructure:</p> <ul style="list-style-type: none"> ▪ Receives the message from the SOAP binding component service ▪ Submits the message for processing to the BPEL process service engine first and the human task service engine second 	" Service Infrastructure " on page 1-5
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.	<p>The BPEL service engine:</p> <ul style="list-style-type: none"> ▪ Receives the message from the Service Infrastructure for processing by the BPEL Loan Process application ▪ Completes processing and returns the message to the Service Infrastructure, which reviews the composite wiring and sends the message onto the human task service engine for manager approval 	" Service Engines and Service Components " on page 1-14
UDDI and MDS	The MDS repository stores descriptions of available services. The UDDI publishes these services to the outside world	The SOAP service used in this composite application is stored in the MDS and can also be published to UDDI.	" Universal Description Discovery and Integration " on page 1-9
Service Archive: Composite (deployment unit)	The deployment unit that describes the composite application	The service archive (SAR) of the composite application to the Service Infrastructure is deployed.	" Deployed Service Archives " on page 1-15

See Also:

- [Section 1.2, "Introduction to Oracle SOA Suite"](#) on page 1-1 for a definition of the SCA assembly model
- [Section 2.2, "Creating an SOA Project in Oracle JDeveloper"](#) on page 2-10 for a tutorial that uses the functionality described in [Table 1–1](#)

1.4.1 Binding Components

Binding components are network protocols and services that connect the SOA platform with the outside 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.
- **References** — Enable messages to be sent from the SOA composite application to external services in the outside world (for example, the same functionality as partner links provide for BPEL processes, but at the higher SOA composite application level). Each BPEL partner link has a corresponding reference at the composite level.

The following binding components are provided:

- **SOAP over HTTP**
- **JCA Adapters** — For integrating services and references with database tables, database queues, file systems, FTP servers, Java Message Services (JMS), IBM WebSphere MQ, BAM servers, or Oracle E-Business Suite applications
- **B2B binding component** — For browsing B2B metadata in the MDS repository and selecting document definitions
- **SDO adapter** — For connecting ADF applications using SDO with the SOA platform
- **RFID** — For providing a bridge to RFID edge sensors and providing business events to the SOA composite application
- **WSIF** — For providing access to custom protocols developed in-house by companies or protocols in the WSIF stack that are not yet compliant with binding component standards. Binding components functionality essentially replaces the WSIF binding functionality of previous releases.

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.

See Also:

- [Chapter 2, "Introduction to the SOA Composite Editor"](#) for details about using service and reference binding components in an SOA project in Oracle JDeveloper
- [Section 1.3.1.4, "Business Events and the Events Delivery Network"](#) on page 1-6
- [Chapter 8, "Business Events and the Event Delivery Network"](#) for details about creating and using business events

1.4.2 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. (provides enterprise service bus (ESB) functionality)
- Complex Event Processing — for continuous queries on event streams

See Also: The following sections for additional details about service components hosted by the service engines:

- [Section 2.1.3, "Introduction to Service Components"](#) on page 2-6
- [Section 2.2.3, "Task 3: Adding a Service Component"](#) on page 2-14

1.4.3 Deployed Service Archives

The SAR is a SOA archive deployment unit that describes the SOA composite application. 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).

1.5 How to Use This Guide

Now that you have a basic understanding of Oracle SOA Suite components and an SOA composite application, it is time to review the parts of this guide. This guide is divided into multiple parts to describe the various Oracle SOA Suite integration technologies. [Table 1–2](#) describes the guide layout.

Table 1–2 Developer's Guide Contents

Part	Description
Part I, "SOA Suite Introduction"	Chapters in this part provide an introduction to the following topics: <ul style="list-style-type: none"> ■ SCA concepts and capabilities ■ SOA Composite Editor concepts and a simple tutorial ■ SOA composite application life cycle and security information

Table 1–2 (Cont.) Developer's Guide Contents

Part	Description
Part II, "Enterprise Service Bus Infrastructure"	<p>Chapters in this part describe how to use the functionality included with Oracle ESB:</p> <ul style="list-style-type: none"> ■ XSLT Mapper ■ Mediator ■ Business events ■ Domain-value maps
Part III, "BPEL Process Service Component"	<p>Chapters in this part provide the following information:</p> <ul style="list-style-type: none"> ■ Introduce you to key BPEL development concepts and associated code samples. These chapters are useful for any developer interested in understanding the underlying functionality of BPEL. ■ Describe how Oracle BPEL Process Manager provides value and ease of use to BPEL functionality
Part IV, "Human Workflow Service Component"	<p>Chapters in this part describe the following:</p> <ul style="list-style-type: none"> ■ How to design and deploy human tasks ■ How to access and act upon human tasks in the Oracle BPM Worklist
Part V, "Oracle Business Activity Monitoring"	<p>Chapters in this part describe how to do the following:</p> <ul style="list-style-type: none"> ■ Create and manage data objects ■ Create and manage external data sources ■ Use alerts ■ Design ADF pages with Oracle BAM data controls
Part VI, "Oracle User Messaging Service"	<p>Chapters in this part describe how to do the following:</p> <ul style="list-style-type: none"> ■ Send and receive messages and develop applications using Oracle Messaging Server ■ Configure the packaged user preferences of the Oracle User Messaging Service ■ Develop drivers
Part VII, "Composite Test Framework"	<p>This part describes how to create, deploy, and run test cases that automate the testing of SOA composite applications.</p>

Introduction to the SOA Composite Editor

This chapter introduces the Service-Oriented Architecture (SOA) Composite Editor. The SOA Composite Editor provides a single environment for designing SOA composite applications.

This chapter contains the following topics:

- [Section 2.1, "Introduction to the SOA Composite Editor"](#)
- [Section 2.2, "Creating an SOA Project in Oracle JDeveloper"](#)

See Also: [Chapter 1, "Service-Oriented Architecture and Oracle SOA Suite"](#) for SCA conceptual details

2.1 Introduction to the SOA Composite Editor

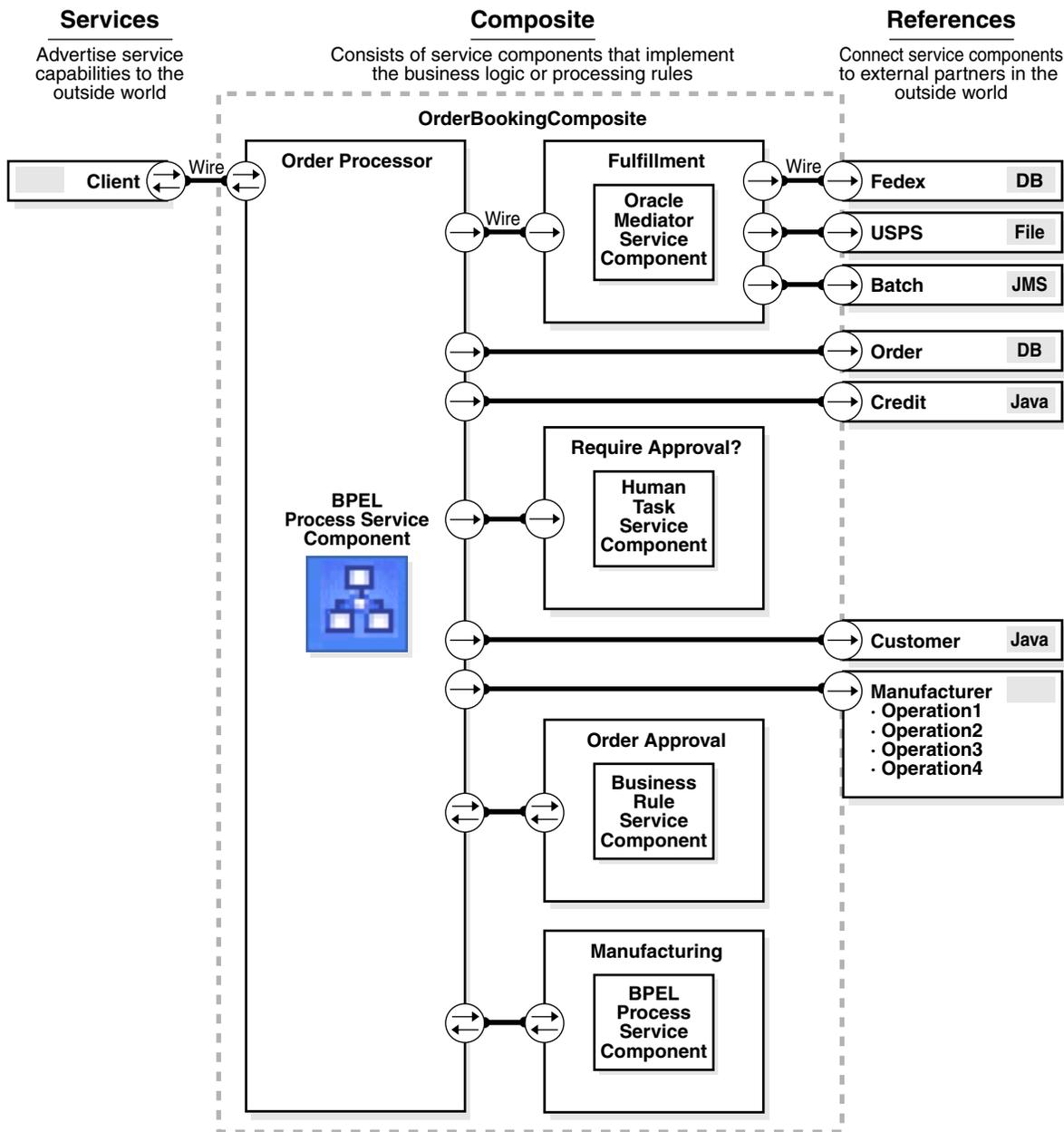
An SOA composite application is typically referred to as a composite application because it consists of numerous services and applications. The SOA Composite Editor enables you to create, edit, and deploy services, but also to assemble them in a composite application, all from a single location. These components are integrated together into one application and communicate with the outside world through binding components such as Web services and JCA adapters.

An SOA composite application can consist of multiple projects (for example, an SOA project, a web project, business components project, and so on). You deploy the application a *single* time into a *single* runtime environment instead of deploying components separately to multiple runtime environments.

Once deployed, you can monitor and manage an SOA composite application from the Oracle Enterprise Manager Fusion Middleware Control Console.

Before designing and deploying an SOA composite application in [Section 2.2, "Creating an SOA Project in Oracle JDeveloper"](#) on page 2-10, you must first understand the key entities of the SOA Composite Editor. [Figure 2-1](#) provides an overview of the entities that you use to design an SOA project in the SOA Composite Editor.

Figure 2-1 Key Entities of SCA Projects in the SOA Composite Editor



The layout shown in [Figure 2-1](#) appears in the SOA Composite Editor during design-time. See [Figure 2-3](#) on page 2-10 for an example of an application in the SOA Composite Editor.

[Table 2-1](#) briefly describes the entities shown in [Figure 2-1](#) and provides references to sections that provide more specific details.

Table 2–1 SOA Composite Editor Key Entities

Element	Description	See Section
Services	Provide the outside world with an entry point to the SOA composite application	Section 2.1.1, "Introduction to Services" on page 2-3
Composite	Describes the entire assembly of the SOA composite application. Included in the composite are service components, which implement the business logic or processing rules.	Section 2.1.2, "Introduction to a Composite and the SCA Descriptor" on page 2-4
Service components	The building blocks of the composite that you construct to implement the business logic or processing rules. Examples of service components are: <ul style="list-style-type: none"> ■ BPEL processes ■ Human tasks ■ Business rules ■ Mediator routing services ■ Complex event processing 	Section 2.1.3, "Introduction to Service Components" on page 2-6
References	Enable messages to be sent from the SOA composite application to external services in the outside world	Section 2.1.4, "Introduction to References" on page 2-8
Wires	Provides the connections between services, service components, and references in an SOA composite application	Section 2.1.5, "Introduction to Wires" on page 2-9

See Also:

- [Chapter 1, "Service-Oriented Architecture and Oracle SOA Suite"](#) for specific descriptions of SCA concepts
- *Oracle Fusion Middleware Administrator's Guide for SOA Suite* for details about managing and monitoring the SOA composite application from Oracle Enterprise Manager Fusion Middleware Control Console

2.1.1 Introduction to Services

Services are a type of binding component that advertise and provide an entry point for messages sent from the outside world to an SOA composite application. Services contain the following:

- A Web Services Description Language (WSDL) file that describes service capabilities
- The binding connectivity that describes the protocols that can communicate with the service (for example, a Web service or a JCA adapter)

Services display on the *left* side (swim lane) of the SOA Composite Editor, as shown in [Figure 2–1](#). You drag and drop a service to the swim lane to invoke the appropriate property editor of the service, as described in [Table 2–2](#).

Table 2–2 Service Editors

Dragging and Dropping This Service...	Invokes This Editor...
Adapters	<p>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, or Oracle E-Business Suite applications.</p> <p>When complete, the service displays in the left swim lane.</p> <p>To make updates later, double-click the service to re-enter the Adapter Configuration Wizard.</p>
Web Service	<p>Create Web Service — Creates a Web (SOAP) invocation service.</p> <p>When complete, the service displays in the left swim lane.</p> <p>To make updates later, double-click the service to display the Update Service window. SOAP is the default network protocol in SOA composite applications.</p>
SDO Service	<p>Create SDO Service — Creates a service data object (SDO) invocation service.</p> <p>When complete, the service displays in the left swim lane.</p> <p>To make updates later, double-click the service to display the Update Service window. The SDO service connects Application Development Framework (ADF) applications using SDO data formats with the SOA application. SDOs simplify the representation of associated data in SOA applications.</p>
B2B	<p>B2B Wizard — Guides you through selection of a document definition.</p>

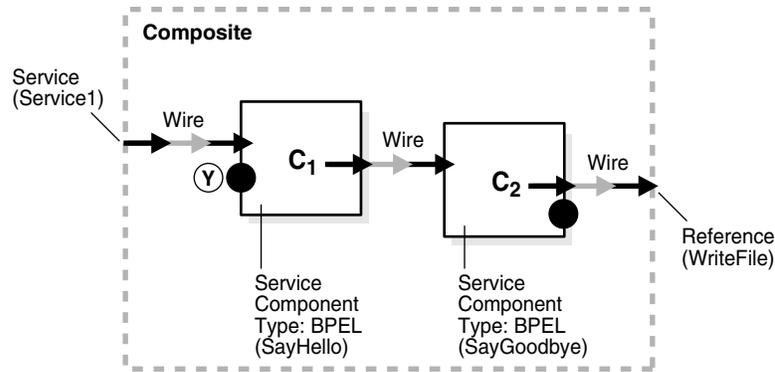
See Also:

- [Section 1.4.3, "Deployed Service Archives"](#) on page 1-15 for additional details about binding component services
- [Figure 2–3](#) on page 2-10 for an example of a service in Oracle JDeveloper
- [Section 2.2.5, "Task 5: Adding a Service"](#) on page 2-18 for an example of creating a service in Oracle JDeveloper

2.1.2 Introduction to a Composite and the SCA Descriptor

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

[Figure 2–2](#) provides an example of a composite that includes two BPEL process service components (named C1 and C2), an inbound service binding component, and an outbound reference binding component.

Figure 2-2 Composite

The SOA composite application is described in the `composite.xml` file. This file is automatically created when you create an SOA project and describes the entire SOA composite. There is one `composite.xml` file per SOA project.

The contents of the `composite.xml` file for the composite shown in [Figure 2-2](#) are described below.

In the initial lines of the file, the service binding component that provides the entry point to the composite is defined. For this example, a binding component Web service named `Service1` is defined. The Web service binding port for delivering messages and policy management details are also defined.

```

<composite name="SayHello">
  . . .
  . . .
  <service name="Service1">
    <interface.wSDL
      interface="http://xmlns.oracle.com/SayHello#wSDL.interface(SayHello)"/>
    <binding.ws port=
      "http://xmlns.oracle.com/SayHello#wSDL.endpoint(Service1/
        SayHello_pt)"/>
  </service>

```

In the following lines, a BPEL process service component named `SayHello` (C1 in [Figure 2-2](#)) is defined:

```

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

```

In the following lines, a second BPEL process service component named `SayGoodbye` (C2 in [Figure 2-2](#)) is defined:

```

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

```

In the following lines, a binding component reference named `WriteFile` is defined. This reference is a JCA file adapter. The reference provides access to the external partner in the outside world.

```

<reference name="WriteFile">
  <interface.wSDL interface=
    "http://xmlns.oracle.com/pcbpel/adapter/file/WriteFile/#wSDL.interface(Write_
      pt)"/>
  <binding.jca config="WriteFile_file.jca"/>

```

```
</reference>
```

In the remaining lines of the file, the communication (or wiring) between service components is described:

- The Web service is wired to the `SayHello` BPEL process service component (represented by C1 in [Figure 2-2](#)). Wiring enables Web service message communication with this specific BPEL process.
- The `SayHello` BPEL process service component (represented by C2 in [Figure 2-2](#)) is wired to the `SayGoodbye` BPEL process service component.
- The `SayGoodbye` BPEL process is wired to the `WriteFile` binding component reference. This is the reference to the external partner in the outside world.

```
<wire>
  <source.uri>Service1</source.uri>
  <target.uri>SayHello/client</target.uri>
</wire>
<wire>
  <source.uri>SayHello</source.uri>
  <target.uri>SayGoodbye/client</target.uri>
</wire>
<wire>
  <source.uri>SayGoodbye/WriteFile1</source.uri>
  <target.uri>WriteFile</target.uri>
</wire>
```

See Also: [Section 1.4.3, "Deployed Service Archives"](#) on page 1-15 for details about message delivery

2.1.3 Introduction to Service Components

Service components are the building blocks of the composite. Each service component (BPEL process, human task, business rules, and mediator routing service) is hosted in its own service engine container. Messages sent to the service engine are targeted at specific service components. For example, a message targeted for a BPEL process is sent to the BPEL service engine.

Service components can implement two types of information:

- Business logic
- Processing rules (declarative rules for processing XML)

Service components display in the canvas workspace (middle area) of an SOA composite application, as shown in [Figure 2-1](#). You drag and drop a service component to the canvas workspace. This action invokes the initial property editor of the service component. After completing the property editor, the service component is created with its initial values. You can specifically design the service component now or at a later time by double-clicking the service component icon. This invokes the specific editor of the service component. [Table 2-3](#) provides an overview.

Table 2–3 Service Component Editors

Dragging and Dropping This Service Component...	Invokes This Editor...	After Initial Creation...
BPEL Process	Create BPEL Process window — Enables you to create a BPEL process that integrates a series of business activities and services into an end-to-end process flow.	Double-click the process to display the BPEL process in Oracle JDeveloper for further designing.
Business Rule	Create Business Rules — Enables you to create a business decision based on rules.	Double-click the business rule to display the Business Rules editor for further designing.
Human Task	Create Human Task — 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.	Double-click the human task to display the Human Task editor for further designing.
Mediator	Create Mediator window — Enables you to define services that perform message and event routing, filtering, and transformations	Double-click the mediator task to display the mediator service in Oracle JDeveloper for further designing.
CEP	Create CEP window — Enables you to write queries to search for patterns in event streams. Oracle Complex Event Processing (CEP) listens on these streams, caches individual, seemingly unrelated events, and tries to correlate them into specific patterns. Each stream is mapped to a business event to which the CEP component subscribes.	Double-click the CEP icon to access an editor for writing queries.

See Also:

- [Section 1.4.2, "Service Engines and Service Components"](#) on page 1-14
- [Section 2.2.3, "Task 3: Adding a Service Component"](#) on page 2-14 for an example of creating a service component in the SOA Composite Editor
- [Section 2.2.4, "Task 4: Editing a Service Component"](#) on page 2-16 for an example of editing a service component in Oracle JDeveloper
- [Chapter 5, "Getting Started with Oracle Mediator"](#) for mediator design details
- [Chapter 11, "Getting Started with Oracle BPEL Process Manager"](#) for Oracle BPEL Process Manager design details
- [Chapter 26, "Designing Human Tasks"](#) for human task design details
- *Oracle Fusion Middleware User's Guide for Oracle Business Rules* for business rule design details

2.1.4 Introduction to References

References are a type of binding component that enables messages to be sent from the SOA composite application to external service partner links in the outside world.

References display on the *right* side (swim lane) of an SOA composite application, as shown in [Figure 2-1](#). You drag and drop a reference to the swim lane to invoke the appropriate property editor of the reference, as shown in [Table 2-3](#).

Table 2-4 Reference Editors

Dragging and Dropping This Reference...	Invokes This Editor...
Adapters	<p>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, or Oracle E-Business Suite applications.</p> <p>When complete, the service displays in the right swim lane.</p> <p>To make updates later, double-click the service to re-enter the Adapter Configuration Wizard.</p>
Web Service	<p>Create Web Service — Creates a Web invocation service.</p> <p>When complete, the service displays in the right swim lane.</p> <p>To make updates later, double-click the service to display the Update Service window.</p>
SDO Service	<p>Create SDO Service — Creates an SDO invocation service.</p> <p>When complete, the service displays in the right swim lane.</p> <p>To make updates later, double-click the service to display the Update Service window.</p>
B2B	<p>B2B Wizard — Guides you through selection of a document definition.</p>

See Also: [Section 11.5, "Partner Link Creation and the SOA Composite Editor"](#) on page 11-15 for details about how creating partner links in the BPEL process of Oracle JDeveloper impacts the services and references in the SOA Composite Editor

See Also:

- [Section 1.4.3, "Deployed Service Archives"](#) on page 1-15
- [Section 2.2.7, "Task 7: Adding a Reference"](#) on page 2-22 for an example of adding a reference in Oracle JDeveloper

2.1.5 Introduction to Wires

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

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

See Also: [Section 2.2.6, "Task 6: Wiring a Service and a Service Component"](#) on page 2-21 for an example of creating wiring in Oracle JDeveloper

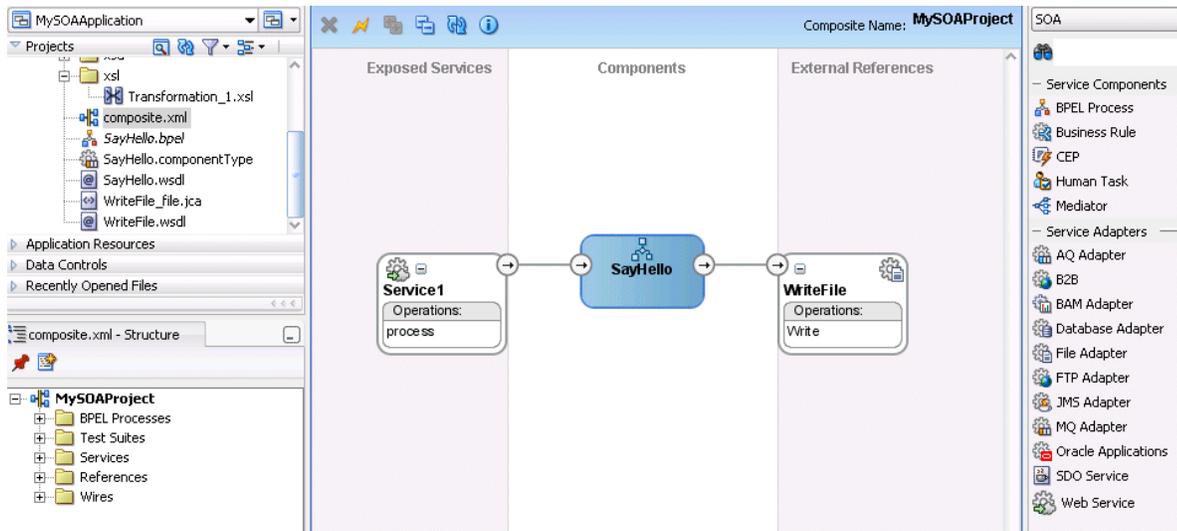
2.1.6 Introduction to SOA Composite Application in Oracle JDeveloper

[Figure 2-3](#) shows an example of an SOA composite application in the SOA Composite Editor of Oracle JDeveloper. Note the use of the entities described in previous sections:

- A Web service named **Service1** displays in the left swim lane.
- A synchronous BPEL process service component named **SayHello** displays in the canvas workspace.
- A file adapter reference named **WriteFile** displays in the right swim lane.
- Wiring between the service, service component, and reference enable message communication.

Subsequent sections of this chapter describe how to create and design a simple SOA composite application.

Figure 2–3 SOA Composite application in the SOA Composite Editor



2.2 Creating an SOA Project in Oracle JDeveloper

Now that the key entities of an SOA composite application have been described, an overview is provided on how to create and design an SOA composite application in Oracle JDeveloper.

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.

This example describes the top-down approach. This section contains the following topics:

- [Section 2.2.1, "Task 1: Creating an Application"](#)
- [Section 2.2.2, "Task 2: Creating an SOA Composite"](#)
- [Section 2.2.3, "Task 3: Adding a Service Component"](#)
- [Section 2.2.4, "Task 4: Editing a Service Component"](#)
- [Section 2.2.5, "Task 5: Adding a Service"](#)
- [Section 2.2.6, "Task 6: Wiring a Service and a Service Component"](#)
- [Section 2.2.7, "Task 7: Adding a Reference"](#)
- [Section 2.2.8, "Task 8: Wiring a Service Component and a Reference"](#)
- [Section 2.2.9, "Task 9: Updating Message Schemas of Components \(Optional\)"](#)

- [Section 2.2.10, "Task 10: Deploying the Project"](#)
- [Section 2.2.11, "Task 11: Testing the SOA Composite Application"](#)

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

2.2.1 Task 1: Creating an Application

You first create an application for the SOA project.

1. Start Oracle JDeveloper.
2. If Oracle JDeveloper is running for the first time, specify the location for Java JDK 1.5.
3. Create a new SOA composite application.

If Oracle JDeveloper...	Then...
Has no open applications	Click New Application in the System Navigator in the upper left.
Has existing applications	Select New > Application from the File main menu or select New Application from the Application menu.

4. Enter the following values:

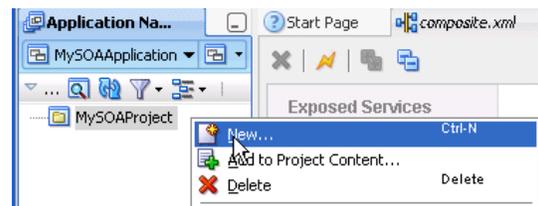
Field	Value
Application Name	Enter an application name (for this example, MySOAApplication is entered).
Application Template	Select No Template [All Technologies] .

5. Accept the default values for all remaining settings, and click **OK**.
The Create Project window appears.
6. Enter a name for the project (for this example, **MySOAProject**), click **OK**.

2.2.2 Task 2: Creating an SOA Composite

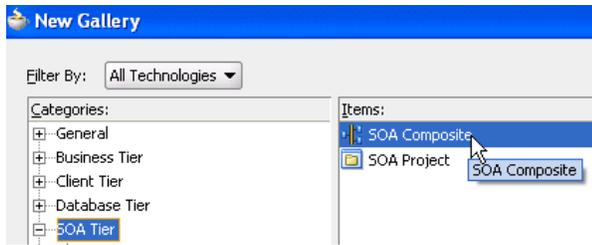
You now create an SOA composite inside the SOA project.

1. Right-click the project name in the **Application Navigator** and select **New**.



The New Gallery window appears.

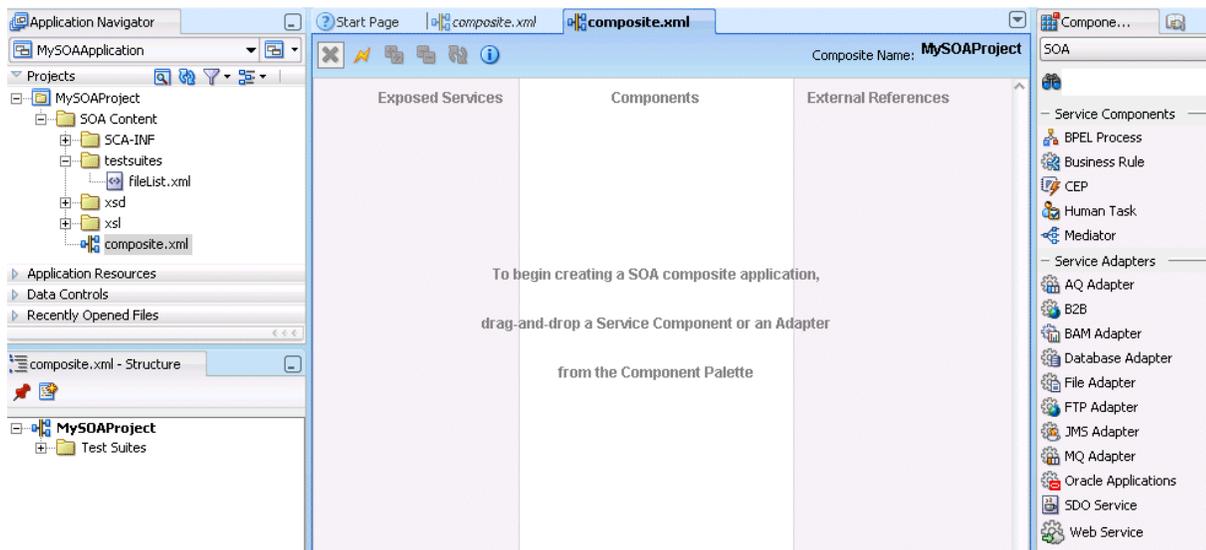
2. Select **SOA Tier** in the **Categories** list.
3. Double-click **SOA Composite** in the **Items** field.



The Create SOA Composite window appears.

4. Select **Empty Composite**.
5. Click **OK**.

The SOA Composite Editor appears.



The SOA Composite Editor consists of the following sections:

Section	Location in Figure	Description
Application Navigator	Left side	<p>Displays the key files for the specific service components included in your SOA project:</p> <ul style="list-style-type: none"> ■ Business rules service component file (<i>rules_name.decs</i>). Additional business rules files display under the Oracle > rules subfolder (<i>rules_name.rules</i>). ■ Mediator service component file (<i>mediator_name.mplan</i>) ■ BPEL process service component files (<i>process_name.bpel</i> and <i>process_name.wsdl</i>) ■ Human task service component files (<i>task_name.task</i> and <i>process_name.decs</i>) ■ A <i>composite.xml</i> file is automatically created when you create a project, and describes the entire composite assembly of services, service components, and references. ■ The <i>componentType</i> file that describes the services and references for each service component ■ Additional subfolders for class files, XSDs, and XSLs.

Section	Location in Figure	Description
Canvas Workspace	Middle	<p>You drag and drop service components into the canvas workspace. When you initially drag and drop these service components, an appropriate property editor is invoked for performing configuration tasks.</p> <p>For all subsequent editing sessions, you double-click these service components to invoke their editors.</p>
Left swim Lane (Exposed Services)	Left side of canvas workspace	The left swim lane is for services providing an entry point to the SOA composite application, such as a Web service or JCA adapters. Services always display in the left swim lane.
Right swim Lane (External References)	Right side of canvas workspace	The right swim lane is for references that send messages to external services in the outside world, such as Web services and JCA adapters. References always display in the right swim lane.
Component Palette	Upper right	<p>Displays the following service components and adapters (binding components):</p> <ul style="list-style-type: none"> ▪ Service Components — Displays the BPEL Process, business rule, human task, mediator service, and CEP components that can be dragged and dropped into the canvas workspace. ▪ Service Adapters — Displays the JCA adapter (AQ, file, FTP, Database, JMS, MQ, Oracle Applications, and BAM), B2B binding component, SDO binding component, and Web service binding component that can be dragged into the left or right swim lanes.
Resource Palette	Lower right	<p>Provides a single window from which to share and access catalogs of resources in the current application and in the metadata. For example, you can access:</p> <ul style="list-style-type: none"> ▪ Shared application metadata such as customer XSDs, WSDLs, business rules, and so on ▪ Seeded metadata that service components in the composite must use such as event definitions, component schemas, component WSDLs, and so on ▪ WSIL browser functionality that uses an HTTP connection or file URL ▪ WSIL browser functionality that uses an Oracle Application Server connection. This functionality is equivalent to the 10.1.3 WSIL browser for discovering deployed services and processes. ▪ UDDI browser functionality <p>If the Resource Catalog does not display, select Resource Palette from the View main menu.</p> <p>You select these resources for the SOA composite application through the SCA Resource Lookup window. This window 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 SCA Resource Lookup window appears. Click Resource Palette at the top of this window to access available resources.</p> <p>See Also: Section 2.2.5, "Task 5: Adding a Service" on page 2-18</p>
Log	Bottom	Displays messages about application compilation, validation, and deployment.
Property Inspector	Bottom	<p>Displays properties for the selected service component, service, or reference.</p> <p>If the Property Inspector does not display, select Property Inspector from the View main menu.</p>

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

2.2.3 Task 3: Adding a Service Component

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

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

Table 2-5 describes the available service components.

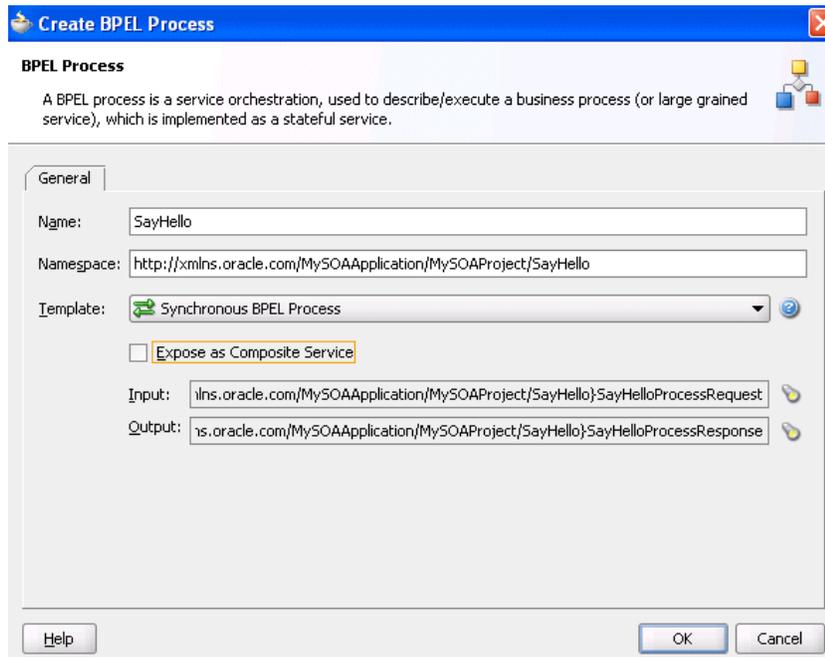
Table 2-5 Starting Service Component Editors

Dragging and Dropping This Service Component...	Starts The...
BPEL Process	Create BPEL Process window — Enables you to create a BPEL process.
Business Rule	Create Business Rules — Enables you to create a business rule.
Human Task	Create Human Task — Enables you to create a human task.
Mediator	Create Mediator window — Enables you to define services from the WSDL and subscribe to events.
CEP	Create CEP window — Enables you to write queries to search for patterns in event streams.

The following example describes the procedures to perform when a BPEL process is dropped into the canvas workspace.

1. Select **SOA** from the **Component Palette**.
2. Drag a **BPEL Process** from the **Service Components** list into the canvas workspace.

The Create BPEL Process window appears.



3. Enter the following details.

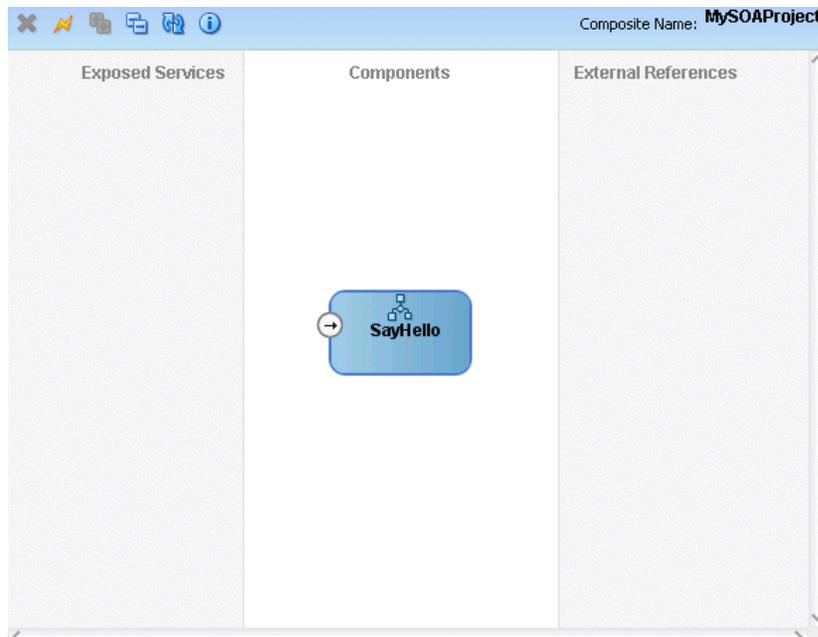
Field	Value
Name	Enter a name (for this example, SayHello is entered).
Namespace	Accept the default value.
Template	Select Synchronous BPEL Process . See Also: The online help for additional details about available templates.
Expose as Composite Service	Deselect this check box. This creates a standalone BPEL process. If you select this check box, a BPEL process and inbound Web service binding component are each created and connected together.

- Note that the **Input** and **Output** fields also appear in the Create BPEL Process window. These fields enable you to select or import specific input and output schemas from the Type Chooser window or SCA Resource Lookup window, 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 SCA Resource Lookup window also provides access to the **Resource Palette**, which provides a single window from which to share and access schemas in multiple applications.

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

The BPEL process displays in the canvas workspace. The single arrow in a circle indicates 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.



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.

2.2.3.1 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 newly created or used from another service component. For example, you can create a human task within the BPEL process service component of Oracle JDeveloper or use a human task that was already created in the SOA Composite Editor or deployed to a server. The reference and the wire is created to the newly created service component.
- The **Resource Palette** can be used to browse for service components defined in the SOA Composite Editor as well as those deployed. A reference and wire is 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 as follows:
 - Right-clicking it and selecting **Delete** from the context menu
 - Highlighting it and selecting **Delete** from the **View** main 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.

2.2.4 Task 4: Editing a Service Component

To model specific details for the service component, you double-click the service component to display the appropriate editor, as described in [Table 2-6](#).

Table 2-6 Starting SOA Service Component Wizards and Windows

Double-Clicking This Service Component...	Displays The...
BPEL Process	BPEL process in Oracle JDeveloper for further designing
Business Rule	Business rule in the Business Rules editor of Oracle JDeveloper for further designing
Human Task	Human task in the Human Task editor of Oracle JDeveloper for further designing
Mediator	Mediator service in Oracle JDeveloper for further designing
CEP	CEP editor for specifying a query in the CQLX file.

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

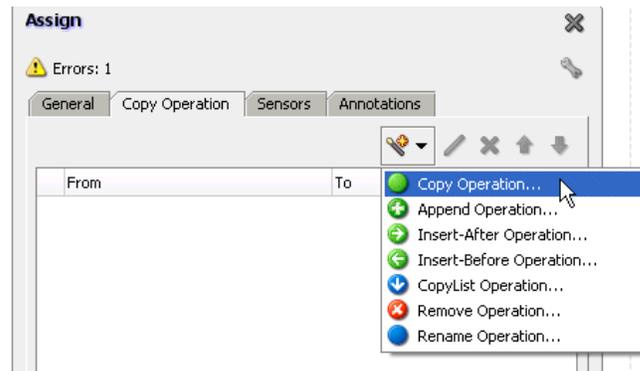
This opens the BPEL process in Oracle JDeveloper.

To return to the SOA Composite Editor from within any service component, double-click `composite.xml` in the **Application Navigator** or single-click `composite.xml` above the canvas workspace.

2. Go to the **Component Palette** in the upper right.



3. Drag and drop an **Assign** activity into the canvas workspace below the **receiveInput** receive activity.
4. Double-click the **Assign** activity.
5. Click the **Copy Operation** tab.
6. Select **Copy Operation** from the drop-down list.



7. Enter the appropriate details. For this example, the following details are entered.

Field	Value
From	<ul style="list-style-type: none"> ▪ Type Expression ▪ Variables <code>concat('Hello',bpws:getVariableData('inputVariable','payload','/client:SayHelloProcessRequest/client:input'))</code> <p>Note: Press Ctrl and then the space bar 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 means 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.</p>
To	<ul style="list-style-type: none"> ▪ Type Variable

Field	Value
Variables	Expand and select Variables > Process > Variables > outputVariable > payload > client:SayHelloProcessResponse > client:result

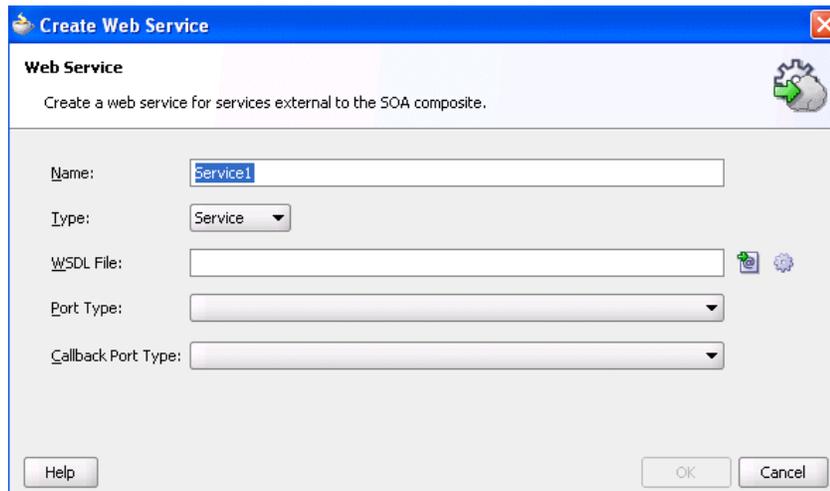
8. Click **OK** to close the Create Copy Operation window and the Assign window.
9. Double-click `composite.xml` in the **Application Navigator** or single-click `composite.xml` above the canvas workspace.
This returns you to the SOA Composite Editor.
10. Select **Save All** from the **File** main menu.

2.2.5 Task 5: Adding a Service

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

Note: This section describes how to manually create a binding component service. You can also automatically create a binding component service by selecting **Expose as Composite Service** when you create a service component. This selection creates an inbound Web service binding component that is automatically connected to your BPEL process or human task service component.

1. Select **SOA** from the **Component Palette**.
2. Drag and drop a **Web Service** to the *left* swim lane.
This invokes the Create Web Service window.

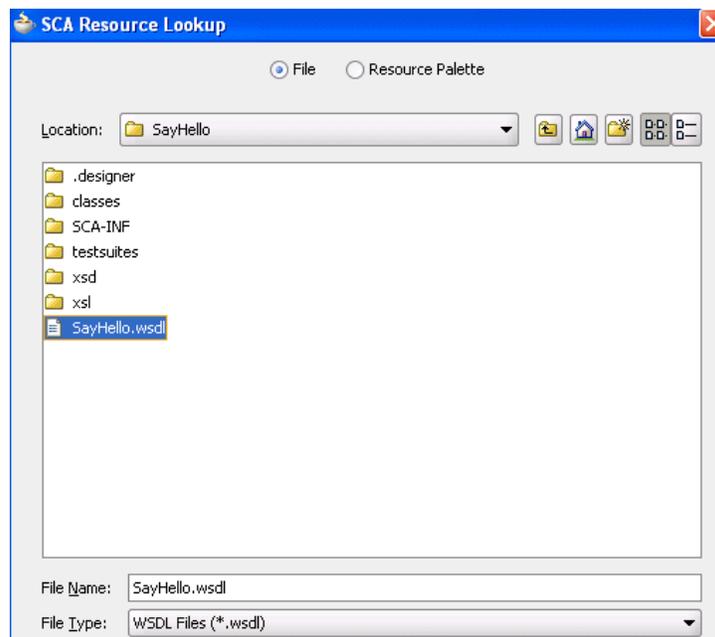


3. Enter the following details:

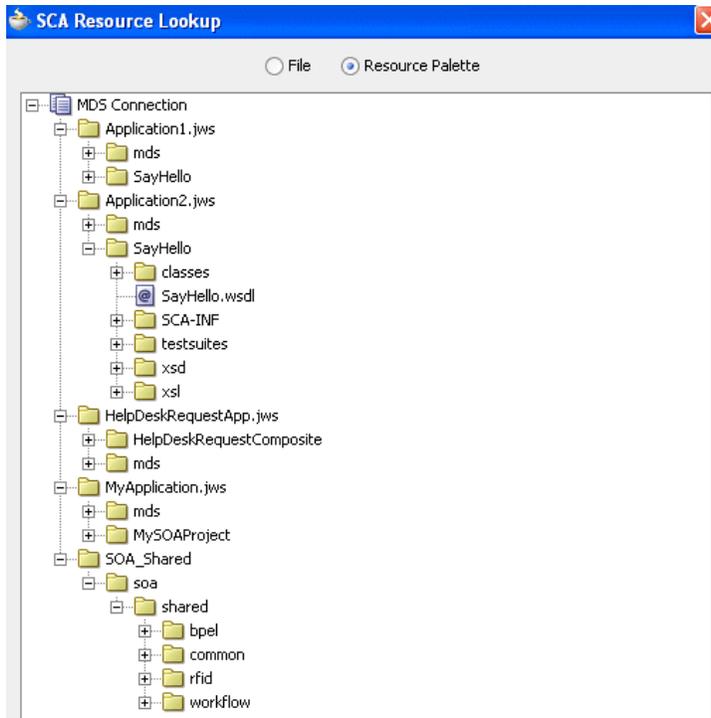
Field	Value
Name	Enter a name for the service (for this example, Service1 is entered).

Field	Value
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) — Creates a Web service to provide an entry point to the SOA composite application. ▪ Reference — Creates a Web service to provide access to an external partner link in the outside world. <p>Since this example describes how to create an entry point to the SOA composite application, Service is selected.</p>

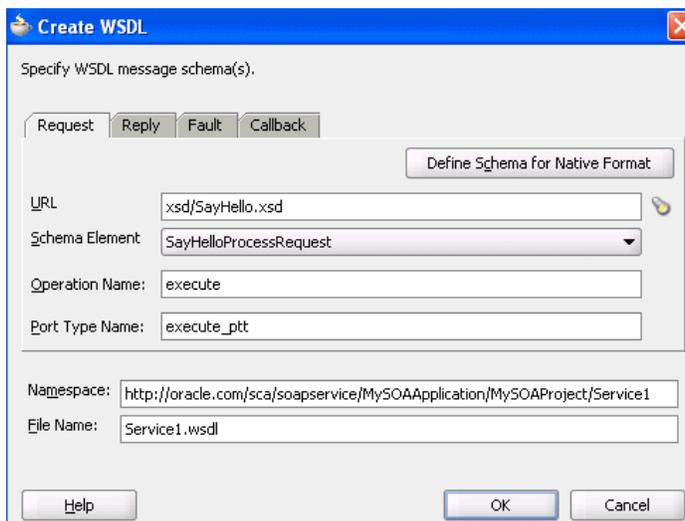
4. Select the WSDL file for the service. There are three methods for selection:
 - a. Click the first icon to the right of the **WSDL File** field, and select an existing WSDL file from the local file system (for this example, **SayHello.wsdl** is selected). Note that the **File** button at the top of the window is automatically selected.



- b. Click the first icon to the right of the **WSDL File** field, and select **Resource Palette** at the top of the window. This enables you to use existing WSDL files from other applications.



- c. Click the second icon to the right of the **WSDL File** field to automatically generate a WSDL file from a schema.

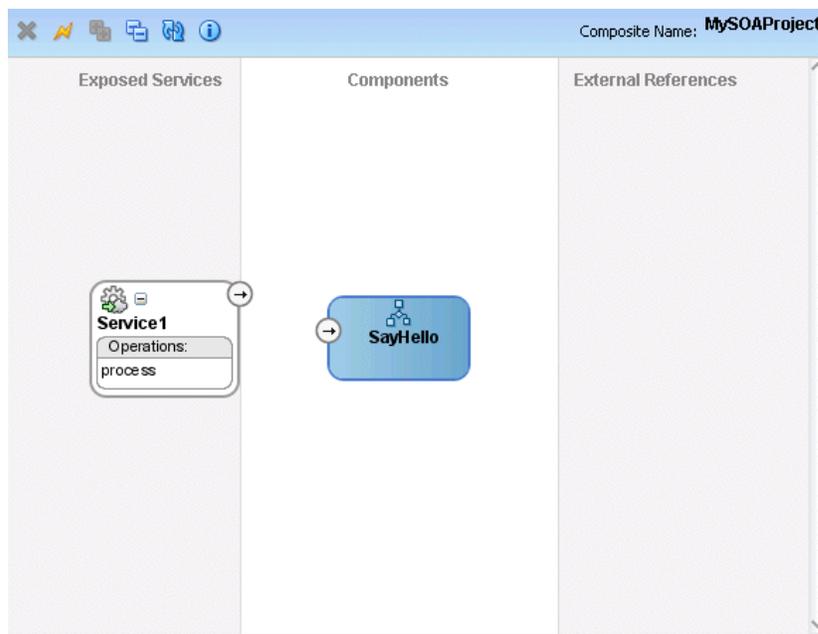


- 5. Click **OK** to return to the Create Web Service window.
- 6. Enter the following additional details:

Field	Value
Port Type	Disabled, since there is only one port type and no choice can be made. This field would be enabled if there was more than one port type from which to choose.
Callback Port Type	Disabled, since this WSDL file is for a synchronous service.

- 7. Click **OK**.

The SOA composite application now looks as follows:



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

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

2.2.5.1 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 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 new partner link that can be connected to a receive or an on-message activity.

Note the following detail about deleting services:

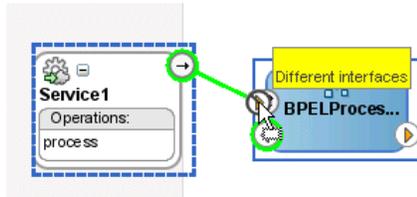
- When a service provided by a service component is deleted, all references to that service component are invalidated and the wires removed.

2.2.6 Task 6: Wiring a Service and a Service Component

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

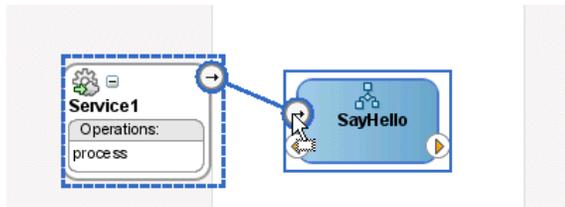
- Since the Web service is an inbound service, a reference handle displays on the right side. Web services that are outbound references do not have a reference handle on the right side.
- You can drag a defined interface to an undefined interface in either direction (reference to service or service to 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 does highlight 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.



The service and reference must match, meaning the interface and the callback must be the same.

1. Drag and drop a wire from the **Service1** reference handle to the **SayHello** BPEL process interface.



2. Create additional service components and wire them together, as needed.

See Also: [Section 2.1.2, "Introduction to a Composite and the SCA Descriptor"](#) on page 2-4 for a description of `composite.xml` file contents

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

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

Note the following detail 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.

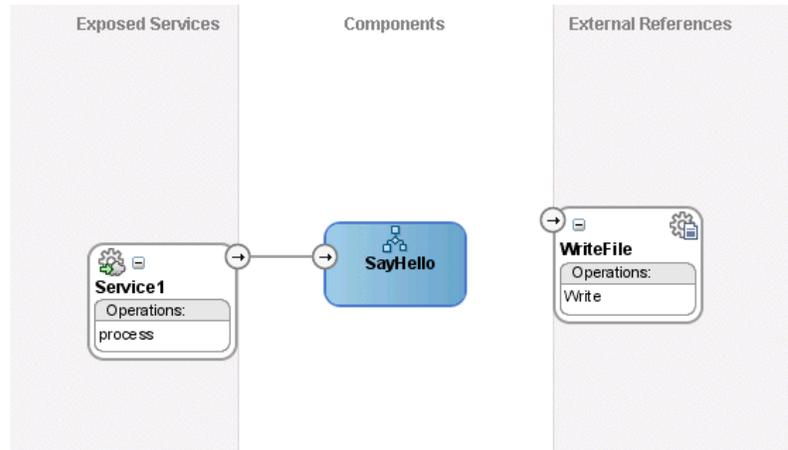
Note the following detail about deleting wires:

- When a wire is deleted, the component's 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).

2.2.7 Task 7: Adding a Reference

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

1. Select **SOA** in the **Component Palette**.
2. Drag and drop a **File Adapter** to the *right* swim lane.
This invokes the Adapter Configuration wizard.
3. Provide appropriate responses to the windows that appear.
When complete, the canvas workspace looks as follows:



See Also: [Section 11.5, "Partner Link Creation and the SOA Composite Editor"](#) on page 11-15 for details about how creating partner links within a BPEL process service component impacts how partner links display in the SOA Composite Editor

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.

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

Note the following details about adding references:

- The only way to add a new reference in the SOA Composite Editor is by wiring the service component to the desired 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.

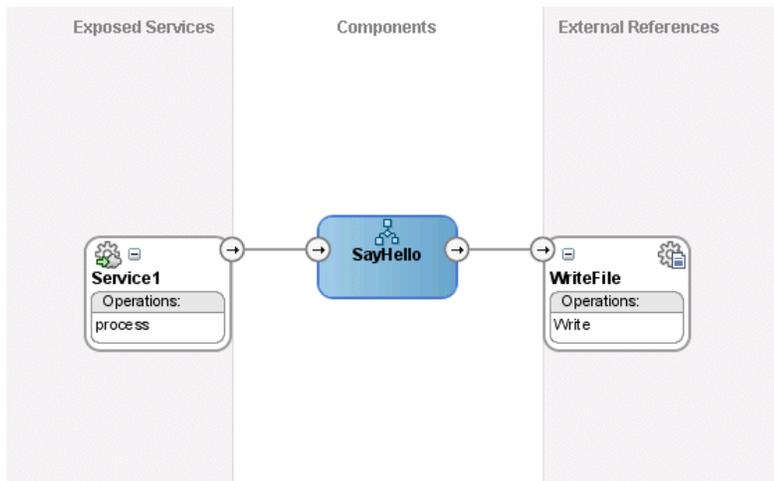
Note the following details about deleting references:

- 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.

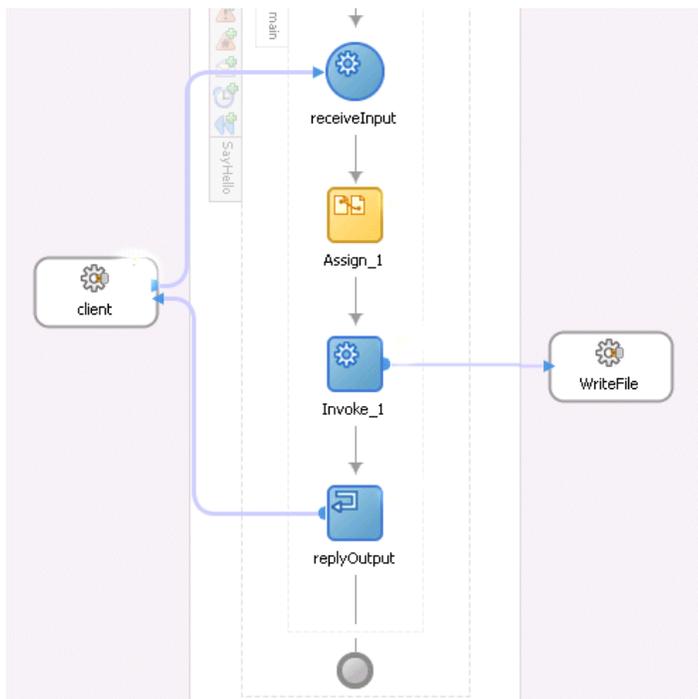
2.2.8 Task 8: Wiring a Service Component and a Reference

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

1. Double-click `composite.xml` in the **Application Navigator** or single-click `composite.xml` above the canvas workspace.
2. Drag and drop a wire from the **SayHello** BPEL process to the **WriteFile** reference.



3. Double-click the **SayHello** BPEL process and note that the **WriteFile** reference displays as a partner link in the right swim lane.

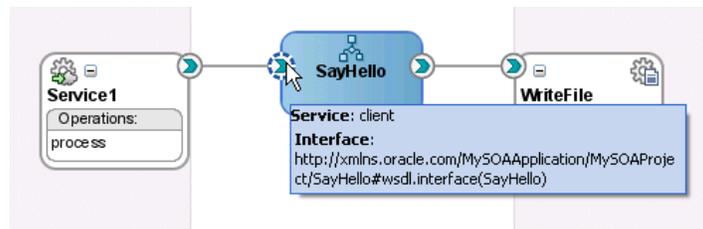


4. Select **Save All** from the **File** main menu.
SOA project design is now complete.

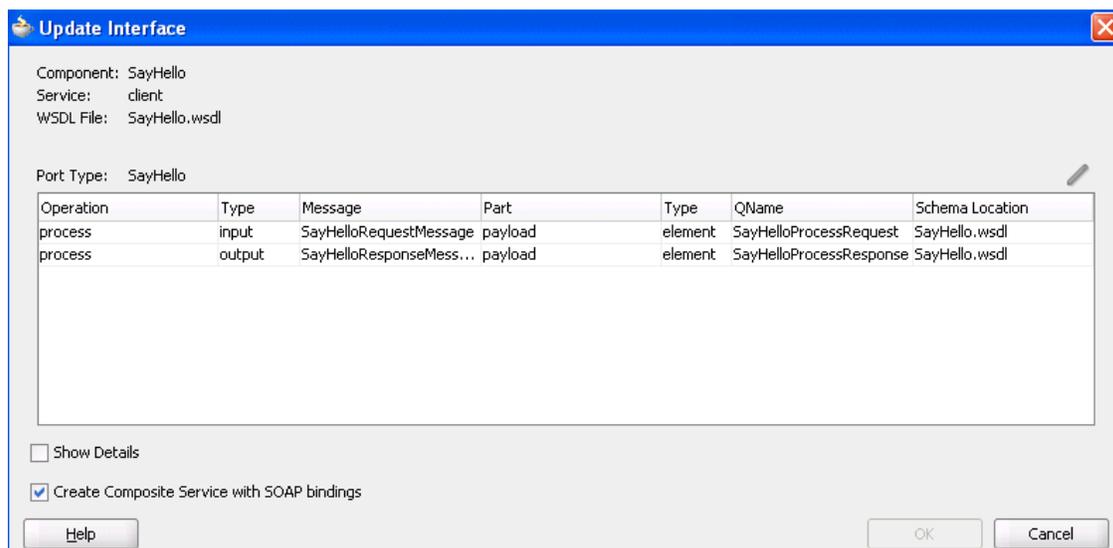
2.2.9 Task 9: Updating Message Schemas of Components (Optional)

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

1. Double-click an interface handle of a component. For this example, the inbound interface handle of the **SayHello** BPEL process service component is selected.



The Update Interface window appears. This window shows all schemas used by the interface's WSDL and enables you to choose a new schema for a selected message part.



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 window appears.

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

2.2.9.1 What You May Need to Know 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 window 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 an interface is wired to another interface, changing one interface does not necessarily change the other interface. If they both use the same WSDL (which is common), then updating one interface automatically updates the other. However, if the wired interfaces use different but compatible WSDLs, then updating one interface does not update the other interface and this can result in a wire between incompatible interfaces. You must then fix the interface on the other side of the wire.
- 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 binding component service or reference), the **Create Composite Service with SOAP bindings** check box appears. This check box provides the same functionality as the **Expose as Composite Service** check box on the BPEL process and human task creation windows. If you check this box and click **OK**, a service and wire are automatically generated. If it is already checked (service already exists) and you unselect it and click **OK**, the service and wire are deleted.

2.2.10 Task 10: Deploying the Project

Deploying the project involves creating an application deployment profile for the SOA composite application. When you deploy an application deployment profile, a service assembly archive (SAR) file is created. The SAR file consists of an EAR file and Oracle metadata. The SAR file replaces the BPEL suitcase archive of previous releases.

See Also: [Section 3.4, "Deploying Applications"](#) on page 3-2 for instructions on creating an application deployment profile

2.2.11 Task 11: Testing the SOA Composite Application

You can run and test instances of deployed SOA composite applications from Oracle Enterprise Manager Grid Control Console.

See Also:

- [Section 3.5, "Testing Applications"](#) on page 3-8 for instructions on testing a SOA composite application
- [Section 3.6, "Troubleshooting Applications"](#) on page 3-8 for instructions on troubleshooting application errors

Life Cycle of an Application

This chapter provides details about the entire life cycle of an application.

This chapter contains the following topics:

- [Section 3.1, "Introduction to Application Life Cycles"](#)
- [Section 3.2, "Methods for Designing Applications"](#)
- [Section 3.3, "Considerations for Collaborative Development"](#)
- [Section 3.4, "Deploying Applications"](#)
- [Section 3.5, "Testing Applications"](#)
- [Section 3.6, "Troubleshooting Applications"](#)

3.1 Introduction to Application Life Cycles

The life cycle of an application can be divided into the following phases:

- Designing the application
- Collaborating or sharing files during design time
- Deploying the application to a server
- Running and testing the application
- Troubleshooting the application

This chapter describes these phases and provides references to other documentation for additional information.

3.2 Methods for Designing Applications

See the *Application Development Framework Developer's Guide* for details about use cases to follow when designing applications with the Application Development Framework (ADF) and Oracle SOA Suite. Most of these use cases fall into three basic patterns:

- Using business events to initiate business processes
- Orchestrating over business logic implemented with ADF, Java, PL/SQL, and SOA composite applications
- Modeling human task service component flows in ADF applications

Guidance on additional, less common use cases that may be useful to applications developers is also provided. For each use case, use of the technology is described and code samples are provided. Details about securing applications are also provided.

See Also: The following documentation for details about securing applications:

- *Oracle Fusion Applications Developer Standards and Guidelines*
- *Oracle Fusion Middleware Application Security Developer's Guide*

3.3 Considerations for Collaborative Development

When you have two separate developer environments collaborating on applications development, it is recommended that you use a source control software system to share files. Two file sharing use cases involving an ADF business component (BC) developer and Oracle SOA Suite developer are described briefly below:

- Use case 1 — Business event file sharing
 1. An ADF BC developer creates a service that uses a business event. This results in the creation of an XSD file and an event definition language (EDL) file for the business event.
 2. The ADF BC developer checks in the XSD and EDL files to a source control software system.
 3. The ADF BC developer advertises the availability of these files.
 4. An Oracle SOA Suite developer checks out the business event XSD and EDL files.
 5. The Oracle SOA Suite developer creates an SOA composite application in which a mediator service component subscribes to this business event.
- Use case 2 — WSDL file sharing
 1. An ADF BC developer creates a service and its associated WSDL file.
 2. The ADF BC developer checks in the WSDL file to a source control software system.
 3. The ADF BC developer advertises the availability of the WSDL file.
 4. An Oracle SOA Suite developer checks out the service WSDL file.
 5. The Oracle SOA Suite developer creates an SOA composite application and imports the WSDL file.
 6. The Oracle SOA Suite developer fully designs the SOA composite application to include a BPEL process that invokes the ADF BC service through a SOAP reference binding component.

See Also: *Application Development Framework Developer's Guide* for specific details about these use cases

3.4 Deploying Applications

You deploy an SOA composite application from Oracle JDeveloper.

3.4.1 Deploying Applications with Oracle JDeveloper

Oracle JDeveloper requires the use of profiles for SOA projects and applications to be deployed. This section describes how to create and deploy profiles with Oracle JDeveloper.

- Section 3.4.1.1, "Task 1: Configuring an SOA Project Deployment Profile (Optional)"
- Section 3.4.1.2, "Task 2: Creating an Application Deployment Profile (Optional)"
- Section 3.4.1.3, "Task 3: Deploying the Application Profile"

3.4.1.1 Task 1: Configuring an SOA Project Deployment Profile (Optional)

The SOA project deployment profile is automatically created as a SOA archive (SAR) profile, and requires no user input. The contents of the SAR profile are deployed to a JAR file.

If you want, you can specify the revision ID value to use. The revision ID value is added to the JAR file name during application deployment and also to the **composite.xml** file of the SOA project.

1. Right-click the SOA project in the **Application Navigator**.
2. Select **Project Properties**.
The Project Properties window appears.
3. Click **Deployment**.
4. Note that **sca_** is automatically prefixed to the composite name of the SAR profile. This prefix is a requirement, and must not be changed.
5. Click **Edit**.
6. View the following values and change if necessary:

Field	Value
Revision ID	Enter the revision ID value to use. This value is added to the JAR file name of the deployed SAR profile name.
Revision Check Box	Select this check box if you want to use the specified revision ID value continually without being prompted again during deployment. If you do not select this check box, you are prompted to specify a revision number when the application profile is deployed. This functionality is similar to the BPEL process versioning feature of previous releases, but at the higher SOA composite application level.

7. Click **OK** to close the SAR Deployment Profile Properties window and the Project Properties window.

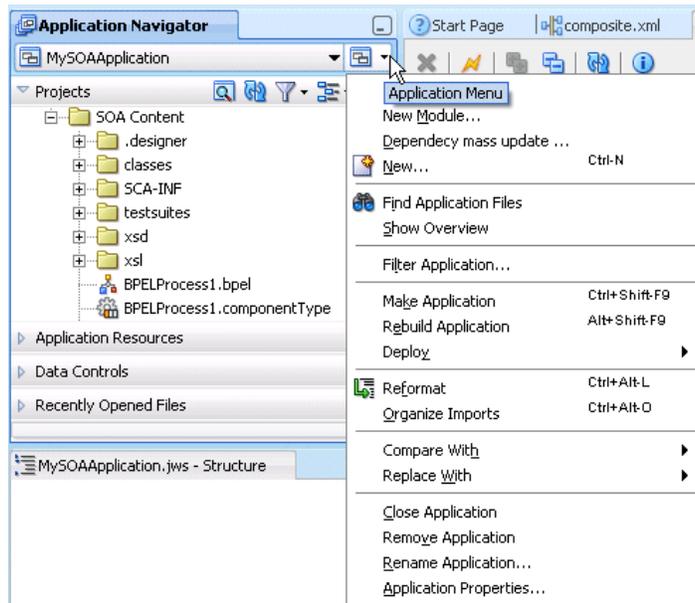
Note: You cannot deploy the SOA project to Oracle Application Server. This is because an enterprise archive (EAR) file is required for deployment. Deployment at the SOA project level only creates a JAR file. Instead, you deploy the application that includes the SOA project.

3.4.1.2 Task 2: Creating an Application Deployment Profile (Optional)

A required deployment profile is automatically created for your application. The application profile includes the JAR files of your SOA projects. Additional profiles are useful for environments in which you want to create multiple deployment profiles for the same application (for example, one profile for a development environment and another profile for a production environment).

If you want, you can manually create a deployment profile.

1. Select the **Application Menu** for the application you want to deploy.



2. Select **Application Properties**.

The Application Properties window appears.

3. Select **Deployment**, and click **New**.

The Create Deployment Profile window appears.

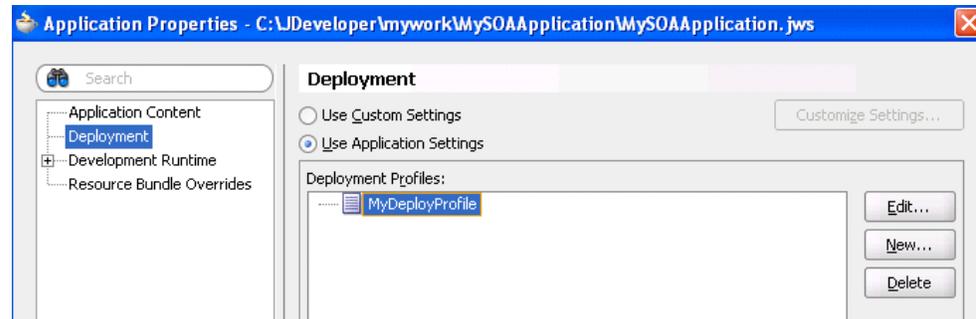
4. Enter the following values:

Field	Value
Archive Type	Select OAR File . This type must be selected. An OAR file is an Oracle Application archive file.
Name	Enter a deployment profile name.

Note: You can change the deployment profile name at a later time by selecting the **General** tab in the OAR Deployment Profile Properties window and changing the name in the **Application name** field.

5. Click **OK**.

The deployment profile displays in the Application Properties window.



6. Click **Edit**.
7. Click **Application Assembly**.

All SOA projects in this application display in the **Java EE Modules** section. For this example, there is only one application (named **sca_MySOAComposite**).

8. Select the SOA projects to include in the deployment profile.



You must select a project. Otherwise, the generated EAR file is empty. The EAR file is what is deployed to Oracle Application Server.

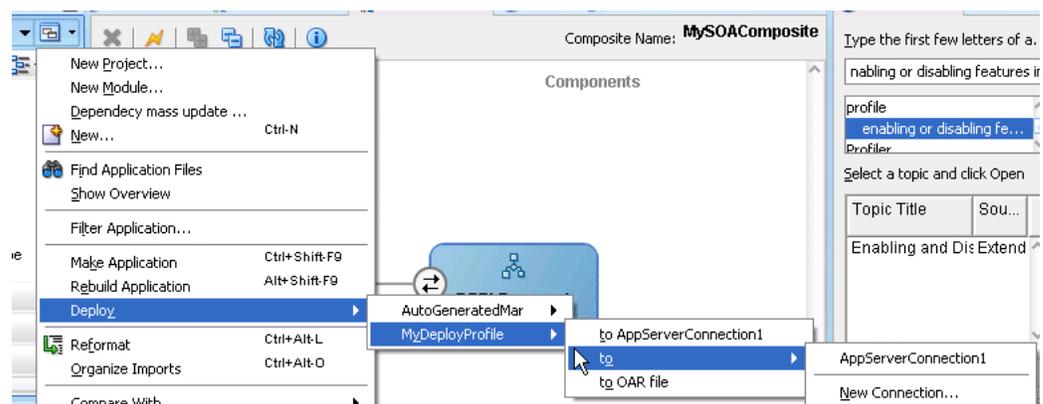
9. Click **OK** to close the OAR Deployment Profile Properties and Application Properties windows.

3.4.1.3 Task 3: Deploying the Application Profile

You now deploy the application profile. The application profile includes the JAR files of the SOA projects you selected in Step 8.

1. Select the **Application Menu** again and select **Deploy > deployment_profile_name**.

The value for *deployment_profile_name* is what you entered in Step 4 on page 3-4.



2. Note that the following deployment options exist:
 - **to *Application_Server_Connection_Name*** — Creates an EAR file of the application deployment profile that includes JAR files of all selected SOA projects and deploys it to Oracle Application Server. An EAR file is required for deployment to the server. To deploy to Oracle Application Server, you must first create a connection to it.
 - **to OAR file** — Creates an EAR file of the application deployment profile that includes JAR files of all selected SOA projects, but does *not* deploy it to Oracle Application Server. This option is useful for environments in which:
 - Oracle Application Server may not be running, but you want to create the EAR file.
 - You want to deploy multiple OAR files to Oracle Application 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.
3. Select the option appropriate for your environment.

If you did not check the **Revision** check box in Step 6 on page 3-3, the Revision ID window appears. Otherwise, go to Step 6.

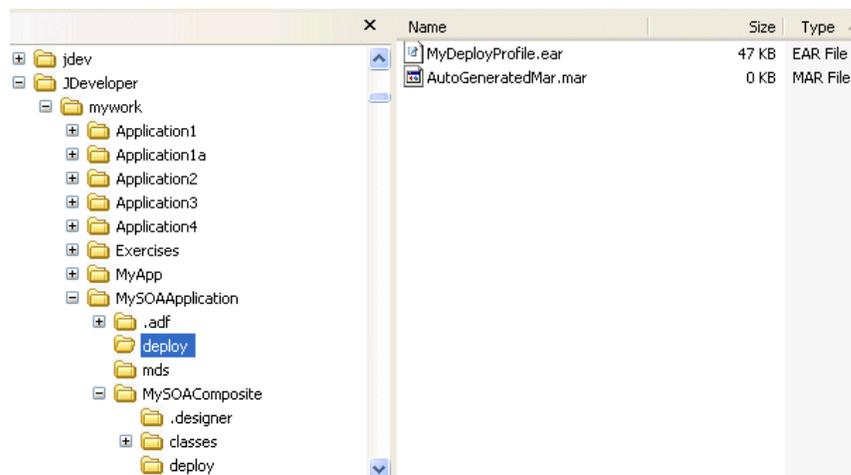


4. Provide a response when prompted for the revision number of the application.
5. Click **OK**.
The Deployment Plan window appears.
6. Perform the following procedures based on the component to which you are deploying:
 - If you are deploying to a standalone OC4J instance, do not modify any parameters.
 - If you are deploying to the Middleware Administration Server (MAS), click **MDS Configuration** and select the Metadata Service (MDS) repository.
7. Click **OK**.
8. If you are deploying an application profile that was previously deployed, you are prompted to undeploy and redeploy the new version of the profile. If you select not to redeploy, deployment does not proceed any further.

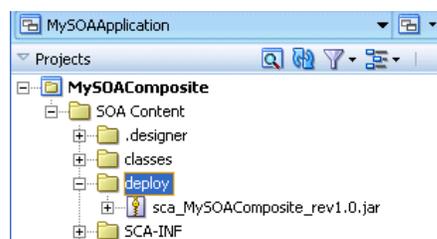
9. View the messages that display in the Log window at the bottom of Oracle JDeveloper.

If deployment is successful, the following files are created:

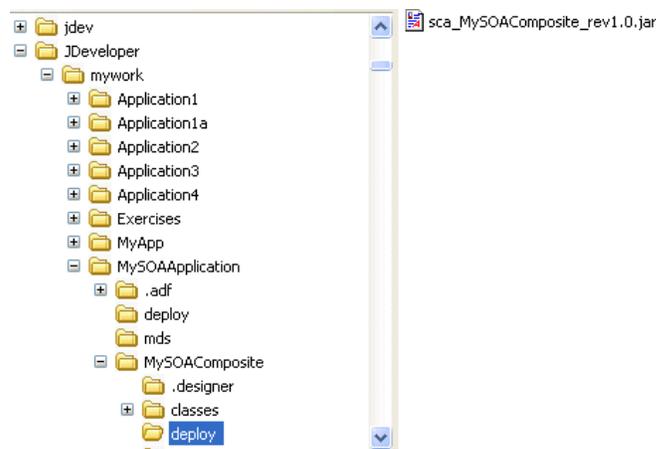
- An EAR file under the *application_name* > **deploy** folder in Windows Explorer with a naming convention of *deployment_profile_name.ear* (for example, **MyDeployProfile.ear**).



- JAR files for each of the SOA projects in the EAR file are created under the **deploy** folder in Oracle JDeveloper with a naming convention of *sca_project_name_revision_number.ear* (for example, **sca_MySOAComposite_rev1.0.jar**).



This JAR file can also be viewed in the *project_name* > **deploy** directory in Windows Explorer:



You are now ready to run your application from Oracle Enterprise Manager Grid Control Console.

If deployment is unsuccessful, view the messages that display in the Log window and take corrective actions. See [Section 3.6.3, "Viewing Deployment Error Files"](#) on page 3-16 for details about files that are provided for debugging deployment errors.

3.5 Testing Applications

You can run and test instances of deployed SOA composite applications from Oracle Enterprise Manager Grid Control Console. This enables you to:

- Manage a composite application
- Initiate an instance of a composite
- Track an instance of a composite
- View detailed component instance audit trails

See Also: *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for instructions on testing applications

3.6 Troubleshooting Applications

Oracle SOA Suite provides the following tools for troubleshooting SOA composite application errors:

- [Section 3.6.1, "Recovering from Faults"](#)
- [Section 3.6.2, "Viewing Log Files"](#)
- [Section 3.6.3, "Viewing Deployment Error Files"](#)

See Also: *Oracle Fusion Middleware Order Demo Developer's Guide for Oracle SOA Suite* for additional information on debugging applications

3.6.1 Recovering from Faults

You can recover from faults that occur in BPEL process and Oracle Mediator service components by defining a fault policy. You define the fault policy in two files during design time. This policy describes how to handle runtime faults. If a fault is defined, it is automatically handled by the fault policy during runtime. If a defined fault results in a condition in which human intervention is the prescribed action, you perform recovery actions from Oracle Enterprise Manager Grid Control Console. Both individual fault recovery and bulk fault recovery are supported.

You can recover from individual faults in Oracle Enterprise Manager Grid Control Console by modifying the following:

- Variable values in BPEL process service components
- Payloads in Oracle Mediator service components

3.6.1.1 Design Time Configuration

You must define fault policy details in the following files prior to SOA composite application deployment. You must create both files manually and include each in the directory path of your SOA project. There is no support for fault policy file creation from Oracle JDeveloper. When you create and deploy your project deployment profile, these files are included in the deployed EAR file.

- [Section 3.6.1.1.1, "fault-policies.xml"](#)
- [Section 3.6.1.1.2, "fault-bindings.xml"](#)

See Also: [Section 3.4, "Deploying Applications"](#) on page 3-2

3.6.1.1.1 fault-policies.xml This file defines fault conditions and their corresponding actions. Each fault policy consists of `condition` and `action` sections. Both sections enable you to define one or more fault conditions and actions, respectively. Each fault condition specifies a particular fault or group of faults, which it attempts to handle, and the corresponding action for it.

```

. . .
. . .
<faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
  name="bpelx:remoteFault">
  <condition>
  <!-- Condition with a filter criteria -->
    <test>$fault.code/code="WSDLReadingError"</test>
    <action ref="ora-terminate"/>
  </condition>
  <condition>
  <!-- Catch all condition for remoteFault -->
    <action ref="ora-java"/>
  </condition>
</faultName>
. . .
. . .

```

All actions defined in the `condition` section must be associated with an action in the `action` section.

Each action has a user-defined unique ID and is based on a system-defined action. Action IDs must be unique across a fault policy. Each action ID can have its own set of attributes. The following syntax shows several attributes, including `retryCount` and `retryInterval`. Available system actions are `retry`, human intervention (recoverable from Oracle Enterprise Manager Grid Control Console), `replay`, `rethrow`, `Java callout` and `abort`. The following syntax shows examples of `retry` and `Java callout`.

```

. . .
  <Actions>
    <Action id="ora-retry">
      <!-- User defined unique id -->
      <retry>
        <!-- System defined action -->
        <retryCount>3</retryCount>
        <retryInterval>2</retryInterval>
        <exponentialBackoff/>
        <!-- System defined action attributes -->
        <retryFailureAction ref="ora-java"/>
        <retrySuccessAction ref="ora-java"/>
      </retry>
    </Action>
    <Action id="ora-java">
      <javaAction
        className="com.oracle.bpel.client.config.faultpolicy.TestJavaAction"
        defaultAction="ora-terminate"
        propertySet="prop-for-billing">
        <returnValue value="ABORT" ref="ora-terminate"/>
        <returnValue value="RETRY" ref="ora-retry"/>
      </javaAction>
    </Action>
  </Actions>

```

```

        <returnValue value="MANUAL" ref="ora-human-intervention"/>
    </javaAction>
</Action>
</Actions>
. . .

```

Other available action IDs include `ora-abort`, `ora-replay-scope`, `ora-human-intervention`, `ora-rethrow-fault`, and `ora-terminate`.

3.6.1.1.2 fault-bindings.xml This file associates the fault policies defined in `fault-policies.xml` with a composite, a particular service component in the composite, a reference binding component, or a port type.

```

<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>
  </component>
  <reference faultPolicy="CRM_ServiceFaults">
    <name>creditRatingService</name>
    <name>BPELSamples-All</name>
    <portType
xmlns:credit="http://services.otn.com">credit:CreditRatingService
    </portType>
    <portType xmlns:db="http://xmlns.oracle.com/pcbpel/adapter/db">
      db:insert_plt
    </portType>
  </reference>
  <reference faultPolicy="test1">
    <name>CreditRating3</name>
  </reference>
</faultPolicyBindings>

```

The order of precedence for fault bindings resolution is reference binding component, port type, service component, and composite, if all are defined in the same `fault-bindings.xml` file. If you have only defined a fault policy on a single composite in the `fault-bindings.xml` file, all service components, port types, and reference binding components of that composite use the same fault policy.

3.6.1.2 BPEL Process and Oracle Mediator Fault Handling Features

Table 3–1 describes the fault recovery actions that you can use in BPEL processes and Oracle Mediator.

Table 3–1 Available Fault Recovery Actions

Action	Supported By BPEL Processes	Supported By Oracle Mediator
Retry	Yes	Yes (with count, exponential break off)
Manual recovery (human intervention)	Yes	Yes
Replay	Yes	No
Rethrow	Yes	No
Java callout	Yes	Yes
Abort	Yes	Yes

Table 3–1 (Cont.) Available Fault Recovery Actions

Action	Supported By BPEL Processes	Supported By Oracle Mediator
Payload-based filter conditions	Yes	No

Note:

- The fault policy framework, if defined, takes precedence over the fault handling features of BPEL processes described in this section.
- Business faults (defined through the WSDL file) in Oracle Mediator are currently handled through routing rules (as was the case in 10.1.3).

3.6.1.3 Recovering from Faults in Oracle Enterprise Manager Grid Control Console

After defining your fault policy in the `fault-policies.xml` and `fault-bindings.xml` files, you perform fault recovery for a SOA composite application from Oracle Enterprise Manager Grid Control Console.

See Also: *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for instructions on recovering from faults

3.6.2 Viewing Log Files

As you test and troubleshoot your application, you can encounter errors. Oracle SOA Suite captures these errors in log files. You can view these log files and use the information provided to take corrective actions.

This section contains the following topics:

- [Section 3.6.2.1, "Log File and Location"](#)
- [Section 3.6.2.2, "Log Level Settings"](#)
- [Section 3.6.2.3, "Setting Logging Levels"](#)
- [Section 3.6.2.4, "Writing Logs to Text Files"](#)
- [Section 3.6.2.5, "Displaying Debug Information"](#)

3.6.2.1 Log File and Location

You set logging properties for Oracle SOA Suite in the `ORACLE_HOME\j2ee\home\config\j2ee-logging.xml` file. Logging information is written to a file named `log.xml` file in a directory that you specify. You can then view the contents of this file to troubleshoot errors. For example, you can search for the time stamp (`msg time`) to indicate when a process was invoked. Or, if an exception error occurred, you can search for the string `exception`.

Information in the `j2ee-logging.xml` file is divided into two main sections:

- The `log_handlers` section — for specifying logging property values such as log file directory location and maximum log file size
- The `loggers` section — for specifying logging properties and logging levels of components

3.6.2.1.1 log_handlers section The `log_handlers` section displays at the top of the `j2ee-logging.xml` file. The BPEL process, human task, business rule, Oracle Mediator, and notification service use the OC4J log handler properties (`oc4j-handler`):

```

. . .
. . .
<log_handlers>

<log_handler name='console-handler' class='java.util.logging.ConsoleHandler'
    formatter='oracle.core.ojdl.logging.SimpleFormatter' />

<log_handler name='oc4j-handler'
    class='oracle.core.ojdl.logging.ODLHandlerFactory'>
    <property name='path' value='../log/oc4j' />
    <property name='maxFileSize' value='10485760' />
    <property name='maxLogSize' value='104857600' />
    <property name='encoding' value='UTF-8' />
    <property name='supplementalAttributes' value='J2EE_APP.name,J2EE_
MODULE.name,WEBSERVICE.name,WEBSERVICE_PORT.name' />
</log_handler>

```

The `path` value specifies the directory in which to write log information. In this example, the directory is `ORACLE_HOME\j2ee\home\log\oc4j`. Logging output is always written to a file named `log.xml` file.

The `maxFileSize` value specifies the maximum size of the log file. The `maxLogSize` value specifies the maximum number of log files. For example, if `maxFileSize` is set to 10 MB and `maxLogSize` is set to 20 MB, you can have two log files that add up to 20 MB in size. When this limit is reached, the files are recycled and restarted at 0 MB.

3.6.2.1.2 loggers section The `loggers` section lets you set the logging properties and levels for individual components. In the following example, Web services auditing logging is set to `NOTIFICATION:1`.

```

<loggers>
. . .
. . .
<logger name='oracle.webservices.management.auditing' level='NOTIFICATION:1'
    useParentHandlers='false'>
    <handler name='oracle-webservices-management-auditing-handler' />
</logger>

```

The `useParentHandlers` property is set by default to `false`, which indicates that the logger does not inherit the log level set for its parent, the root logger. This setting causes the logger to use its own handler and create its own log file. If `useParentHandlers` is set to `true`, the logger uses the parent's log file.

3.6.2.2 Log Level Settings

Table 3–2 describes the values to which you can set log properties for the BPEL process, human task, business rule, Oracle Mediator, and notification service. Values are listed from highest to lowest levels. For example, if you specify a logging level of `NOTIFICATION:1`, this includes logging for `WARNING:1`, `ERROR:1`, and `INCIDENT_ERROR:1`.

Table 3–2 Logging Levels

Logging Level	This Selection...	Performance Impact
INCIDENT_ ERROR: 1	Logs serious problems caused by unknown factors. Contact Oracle Support Services to resolve this error.	No performance impact. This is the highest setting.
ERROR: 1	Logs serious problems that require immediate attention from a system administrator. These errors are not caused by a problem in Oracle SOA Suite.	No performance impact.
WARNING: 1	Logs potential problems that must be reviewed by a system administrator.	No performance impact.
NOTIFICATION: 1 (the default setting)	Logs major lifecycle events such as the activation or deactivation of a primary subcomponent or feature.	No performance impact.
NOTIFICATION: 16	Provides a finer level of logging granularity for reporting normal events.	Minimal performance impact. You can enable this level broadly in a production environment without significantly impacting performance.
TRACE: 1	Logs trace or debugging information for events that are meaningful to end users of Oracle SOA Suite, such as public API entry and exit points.	Small performance impact. You can enable this level broadly on occasion in a production environment to debug issues by writing to a circular memory buffer (memory handler). Enabling logging at this level may have a small performance impact, but does not make Oracle SOA Suite unusable.
TRACE: 16	Logs detailed trace or debugging information that assists Oracle Support Services in diagnosing problems with a particular subsystem.	High. Do not enable this level in a production environment, except for special situations in which to debug issues. This level is not to be enabled broadly, but rather for a few specific subsystems.
TRACE: 32	Logs very detailed trace or debug information typically intended for a developer working on Oracle SOA Suite with enough knowledge about the implementation of the subsystem that generates the message.	Very high. Do not enable this level in a production environment. It is intended for debugging Oracle SOA Suite in a test or development environment. This is the lowest setting.

3.6.2.3 Setting Logging Levels

This section describes how to set logging levels for the BPEL process, human task, business rule, notification service, and Oracle Mediator:

- [Section 3.6.2.3.1, "Setting BPEL Process Logging"](#)
- [Section 3.6.2.3.2, "Setting Human Task, Business Rule, and Notification Service Logging"](#)
- [Section 3.6.2.3.3, "Setting Oracle Mediator Logging"](#)

3.6.2.3.1 Setting BPEL Process Logging BPEL process logging is enabled by specifying the properties described in [Table 3–3](#) on page 3-14. For example, you add the following block of code for the `oracle.orabpel.system` and `oracle.orabpel.domain` properties to the `loggers` section and set their logging levels to `NOTIFICATION:1` as follows:

```
<loggers>
. . .

  <logger name="oracle.orabpel.system" level="NOTIFICATION:1"
    useParentHandlers="false">
    <handler name="oc4j-handler"/>
    <handler name="console-handler"/>
  </logger>

  <logger name="oracle.orabpel.domain" level="NOTIFICATION:1"
useParentHandlers="false">
    <handler name="oc4j-handler"/>
    <handler name="console-handler"/>
  </logger>
```

The `console-handler` setting for both of these properties enables you to also output logging information to a DOS window or Linux shell.

[Table 3–3](#) describes the BPEL process properties that you can add to the `loggers` section.

Table 3–3 Property Settings

Logger Name	Description
<code>oracle.orabpel.system</code>	System level logging
<code>oracle.orabpel.domain.activation</code>	Activation agent logging (inbound adapters)
<code>oracle.orabpel.domain</code>	XML message logging
<code>oracle.orabpel.domain.agents</code>	Internal agent framework logging (for example, expiration agents)
<code>oracle.orabpel.domain.bpel</code>	BPEL process logging. If enabled, each executed BPEL activity logs messages.
<code>oracle.orabpel.domain.data</code>	Persistence and dehydration layer logging
<code>oracle.orabpel.domain.delivery</code>	Delivery service and manager logging; this logger is responsible for callbacks and initiating delivery
<code>oracle.orabpel.domain.deployment</code>	Deployment logging
<code>oracle.orabpel.domain.dispatch</code>	Asynchronous message logging
<code>oracle.orabpel.domain.sensor</code>	Sensor publisher layer logging
<code>oracle.orabpel.domain.translation</code>	Adapter translation layer logging (the transformation between adapter protocol and inbound XML documents)
<code>oracle.orabpel.domain.ws</code>	Communication-related logging (for example, SOAP and adapters)
<code>oracle.orabpel.domain.xml</code>	XML processing and transformation, XPath, and XML documents (BPEL variables) logging

Note: Starting with this release, there is no domain level configuration logging. Domains no longer exist in Oracle BPEL Process Manager.

3.6.2.3.2 Setting Human Task, Business Rule, and Notification Service Logging Human task, business rule, and notification service logging is enabled by adding the following block of code for the `oracle.orabpel.services` property. In the following example, the logging level for this property is set to `NOTIFICATION:1`:

```
<loggers>
    .....
    <logger name="oracle.bpel.services" level="NOTIFICATION:1"
        useParentHandlers="false">
        <handler name="oc4j-handler"/>
        <handler name="console-handler"/>
    </logger>
</loggers>
</logging_configuration>
```

3.6.2.3.3 Setting Oracle Mediator Logging Oracle Mediator logging is enabled by adding the following block of code for the `oracle.tip.mediator.dispatch` property. In the following example, the logging level for this property is set to `NOTIFICATION:1`:

```
<loggers>
    .....
    <logger name='oracle.tip.mediator.dispatch' level='NOTIFICATION:1'
        useParentHandlers='false'>
        <handler name='oc4j-handler' />
        <handler name="console-handler" />
    </logger>
```

3.6.2.4 Writing Logs to Text Files

By default, log file information is output in XML format. If you want log file output to display in text format, set the `format` property to `ODL-Text` in the `j2ee-logging.xml` file. For example, set it for the BPEL process component as follows:

```
<log_handlers>
    <log_handler name='oracle-orabpel-handler'
        class='oracle.core.ojdl.logging.ODLHandlerFactory'>
        <property name='path' value='../log/oc4j/bpel' />
        <property name="maxFileSize" value="10485760" />
        <property name="maxLogSize" value="104857600" />
        <property name="encoding" value="UTF-8" />
        <property name='format' value='ODL-Text' />
        <property name="supplementalAttributes" value="J2EE_APP.name,J2EE_
            MODULE.name,WEBSERVICE.name,WEBSERVICE_PORT.name" />
    </log_handler>
```

3.6.2.5 Displaying Debug Information

By default, debug information is not output to the console window (for example, a DOS window or a Linux shell). To display this information in a console window, set the console handler level to `TRACE` in the `j2ee-logging.xml` file as follows:

```
<loggers>
    . . .
    . . . <log_handler name='console-handler-orabpel'
```

```
class='java.util.logging.ConsoleHandler'  
formatter='oracle.core.ojdl.logging.SimpleFormatter' level='TRACE:1' />
```

3.6.3 Viewing Deployment Error Files

If you receive errors during application compilation and deployment, two additional debugging files are created in Oracle JDeveloper by default in the **SOA Content > classes** directory. These files are for advanced users.

- `scac.log.txt` — Shows compilation errors for the entire SOA composite application
- `scac_out.xml` — Shows compilation messages that also display in the Log window.

Note: You can change the default output directory location by selecting **Tools > Project Properties**, and changing the directory path in the **Output Directory** field.

Enterprise Service Bus Infrastructure

This part describes the components that comprise the Enterprise Service Bus Infrastructure.

This part contains the following chapters:

- XSLT Mapper
 - [Chapter 4, "XSLT Mapper and Transformations"](#)
- Oracle Mediator
 - [Chapter 5, "Getting Started with Oracle Mediator"](#)
 - [Chapter 6, "Creating Routing Rules"](#)
 - [Chapter 7, "Oracle Mediator Error Handling"](#)
- Oracle Business Events
 - [Chapter 8, "Business Events and the Event Delivery Network"](#)
- Domain-Value Maps
 - [Chapter 9, "Working with Domain Value Maps"](#)
- Cross References
 - [Chapter 10, "Working with Cross References"](#)

XSLT Mapper and Transformations

This chapter describes features of 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. This chapter provides step-by-step instructions for mapping a sample purchase order schema to an invoice schema.

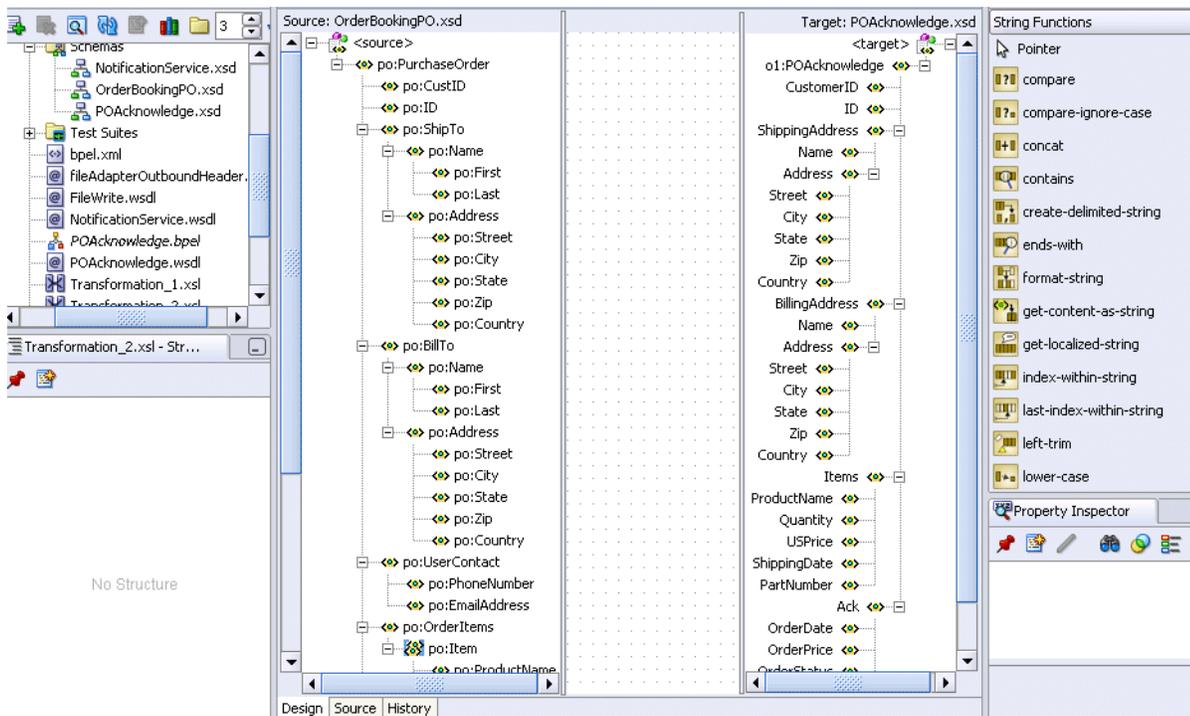
This chapter includes the following sections:

- [Section 4.1, "Introduction to the XSLT Mapper"](#)
- [Section 4.2, "Creating an XSL Map File"](#)
- [Section 4.3, "Using the XSLT Mapper"](#)
- [Section 4.4, "Testing the Map"](#)
- [Section 4.5, "Use Case for Transformation"](#)

4.1 Introduction to the XSLT Mapper

You use the XSLT Mapper transformation tool to create the contents of a map file. [Figure 4–1](#) shows the layout of the XSLT Mapper.

Figure 4–1 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 Figure 4–1 (for example, type, cardinality, and so on). The **Component Palette** in the upper right of Figure 4–1 is the container for all functions provided by the XSLT Mapper. The mapper pane or canvas 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 Mapper 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 mapper pane.

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
- **All Pages** — Provides a view of all functions together 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 mapper pane or canvas in the middle

Right-click each of the three separate panels to see what the context menus look like. A full set of **Undo Auto Map**, **Redo**, **Delete**, and **Delete All** functions are also available.

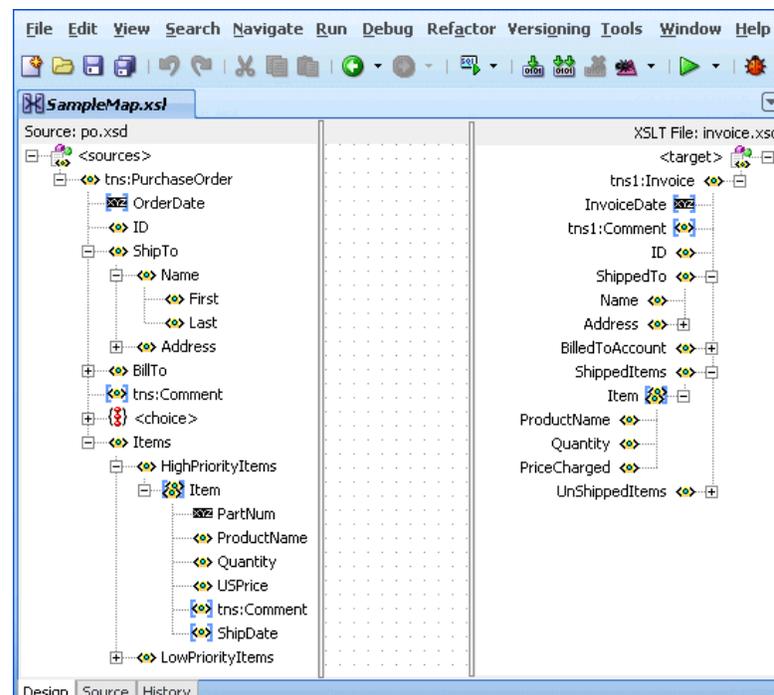
By default, design view shows all defined prefixes for all nodes in the source and target trees. You can elect not to display prefixes by selecting **Hide Prefixes** from context menu on the center canvas. Once prefixes are hidden, select **Show Prefixes** to display them again.

4.1.1 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.

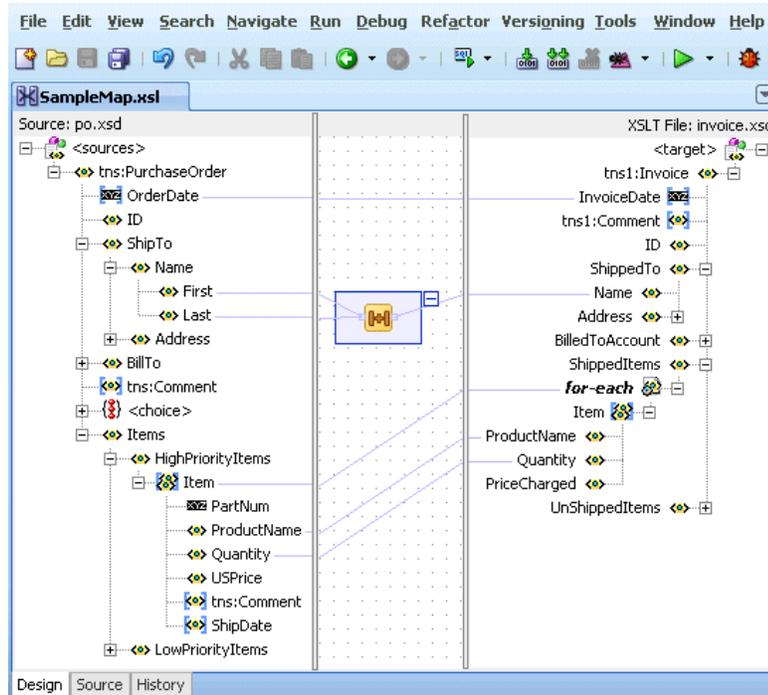
After creating an initial map as described in [Section 4.2, "Creating an XSL Map File"](#) on page 4-6, the mapper displays a graphical representation of the source and target schemas, as shown in [Figure 4-2](#).

Figure 4-2 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 4-3](#).

Figure 4-3 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 4-3](#). This results in a **value-of** statement in the XSLT, as shown in [Example 4-1](#).

Example 4-1 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 4-3](#) are concatenated together 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 4-2](#):

Example 4-2 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 4-3](#). 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 4-3](#) is generated:

Example 4-3 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 4-3](#) 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 below 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 need to 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 mapper does not modify it.

The entire XSLT map generated for this example is shown in [Example 4-4](#):

Example 4-4 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.

4.1.2 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"
```

The workaround is to give each element a unique name.

4.2 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:

- [Section 4.2.1, "How to Create an XSL Map File in Oracle BPEL Process Manager"](#)
- [Section 4.2.2, "How to Create an XSL Map File from Imported Source and Target Schema Files in Oracle BPEL Process Manager"](#)
- [Section 4.2.3, "How to Create an XSL Map File in Oracle Mediator"](#)

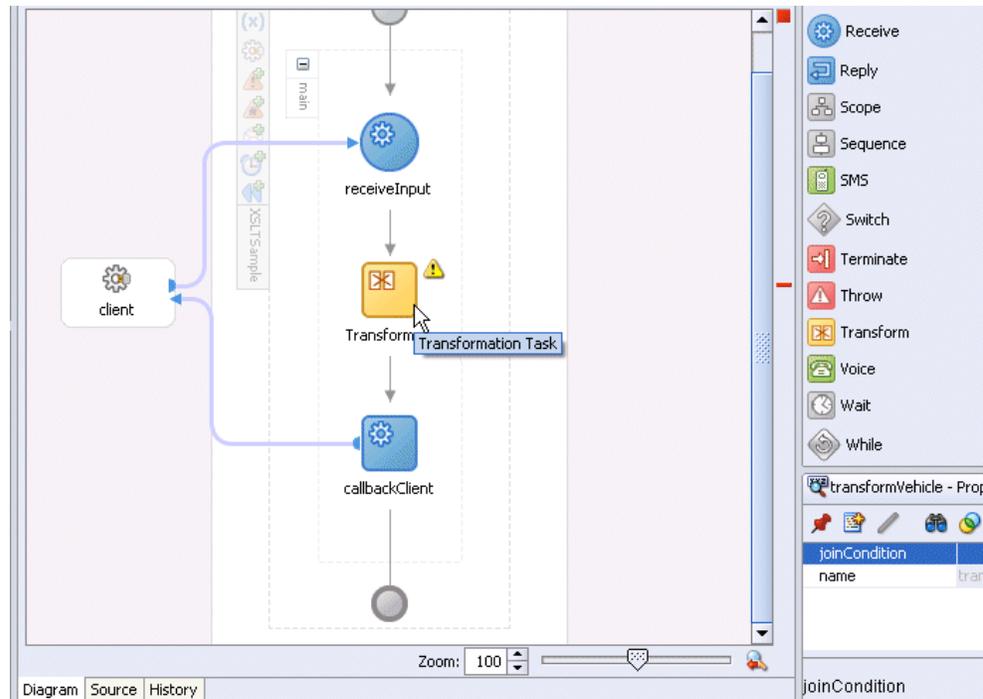
Note: You can also create an XSL map file from an XSL stylesheet. Click **New > General > XML > XSL Map From XSL Stylesheet** from the **File** main menu in Oracle JDeveloper.

4.2.1 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.

1. Drag and drop a **transform** activity from the **Component Palette** into your BPEL process diagram. [Figure 4–4](#) provides an example.

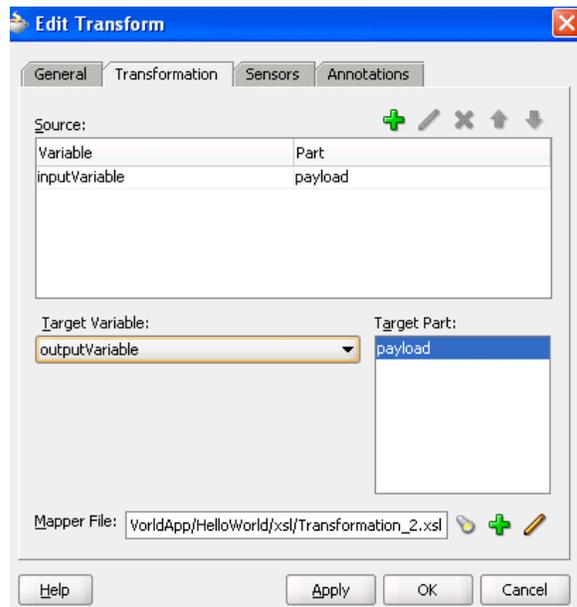
Figure 4-4 Transform Activity



2. Double-click the **transform** activity.

The Transform dialog shown in Figure 4-5 appears.

Figure 4-5 Transform Dialog



3. Specify the following information:

- Source variable from which to map elements
- Source part of the variable (for example, a payload schema consisting of a purchase order request) from which to map. You can select multiple source

variables. These variables can be in a single schema file or in multiple schema files.

- Target variable to which to map elements
 - Target part of the variable (for example, a payload schema consisting of an invoice) to which to map
4. Specify a map file name or accept the default name in the **Mapper File** field. The map file is the file in which you create your mappings using the XSLT Mapper transformation tool.
 5. Click the magic wand icon (second icon) to create a new mapping. If the file already exists, click the note pad icon (third icon) to edit the mapping.
The XSLT Mapper appears.
 6. Go to [Section 4.1, "Introduction to the XSLT Mapper"](#) on page 4-1 for an overview of using the XSLT Mapper.

4.2.2 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 mapper pane to create a new XSL file named `xform.xslt.xml`, even though your intention was to use the existing `xform.xslt` file. A `.xml` extension is appended to *any* file that does not already have a `.xml` extension, and you must create the mappings in the new file. As a workaround, ensure that your files first have an extension of `.xml`. If the XSL file has an extension of `.xslt`, then rename it to `.xml`.

The following steps provide a high level overview of how to create an XSL map in Oracle BPEL Process Manager using a `po.xsd` and `invoice.xsd` file.

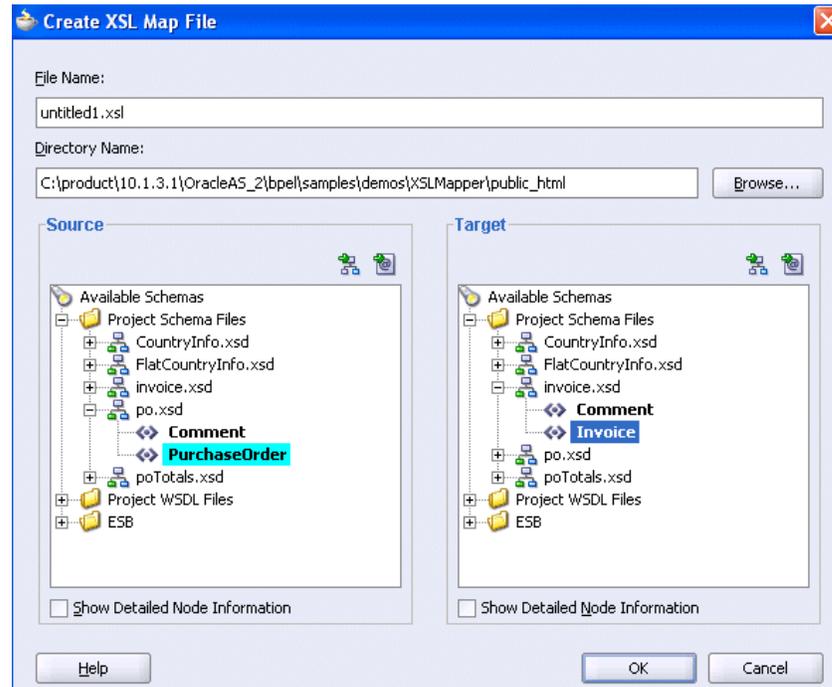
1. In Oracle JDeveloper, select the application project in which you want to create the new XSL map.
2. Import the `po.xsd` and `invoice.xsd` files into the project (for example, by right-clicking **Schemas** and selecting **Import Schemas** in the **Structure** section of Oracle JDeveloper).
3. Right-click the selected project and select **New**.
The New Gallery window 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 window appears. This window enables you to create an XSL map file that maps a root element of a source schema file or WSDL file to a root element of a target schema file or WSDL file.

- 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).

6. Enter a name for the XSL map file in the **File Name** field.
7. Under **Source**, expand **Project Schema Files > po.xsd > PurchaseOrder** as the root element for the source.
8. Under **Target**, expand **Project Schema files > invoice.xsd > Invoice** as the root element for the target. [Figure 4-6](#) provides an example.

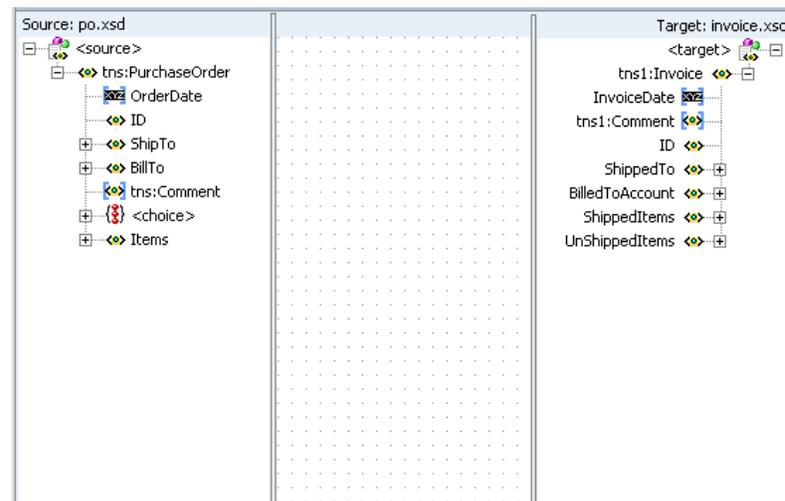
Figure 4-6 Expanded Target Section



9. Click **OK**.

A new XSL map is created, as shown in [Figure 4-7](#).

Figure 4-7 New XSL Map



10. Save and close the file now or begin to design your transformation. Information on using the XSLT Mapper is provided in [Section 4.1, "Introduction to the XSLT Mapper"](#) on page 4-1.
11. Drag and drop a **transform** activity from the **Component Palette** into your BPEL process.
12. Double-click the **transform** activity.
13. Specify the following information:
 - Source variable from which to map elements
 - Source part of the variable (for example, a payload schema consisting of a purchase order request) from which to map
 - Target variable to which to map elements
 - Target part of the variable (for example, a payload schema consisting of an invoice) to which to map
14. Click the **flashlight** icon (first icon) to the right of the **Mapper File** field to browse for the map file name you specified in Step 6.
15. Click **Open**.
16. Click **OK**.

The XSLT Mapper displays your XSL map file.
17. Go to [Section 4.1, "Introduction to the XSLT Mapper"](#) on page 4-1 for an overview of using the XSLT Mapper.

4.2.3 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.

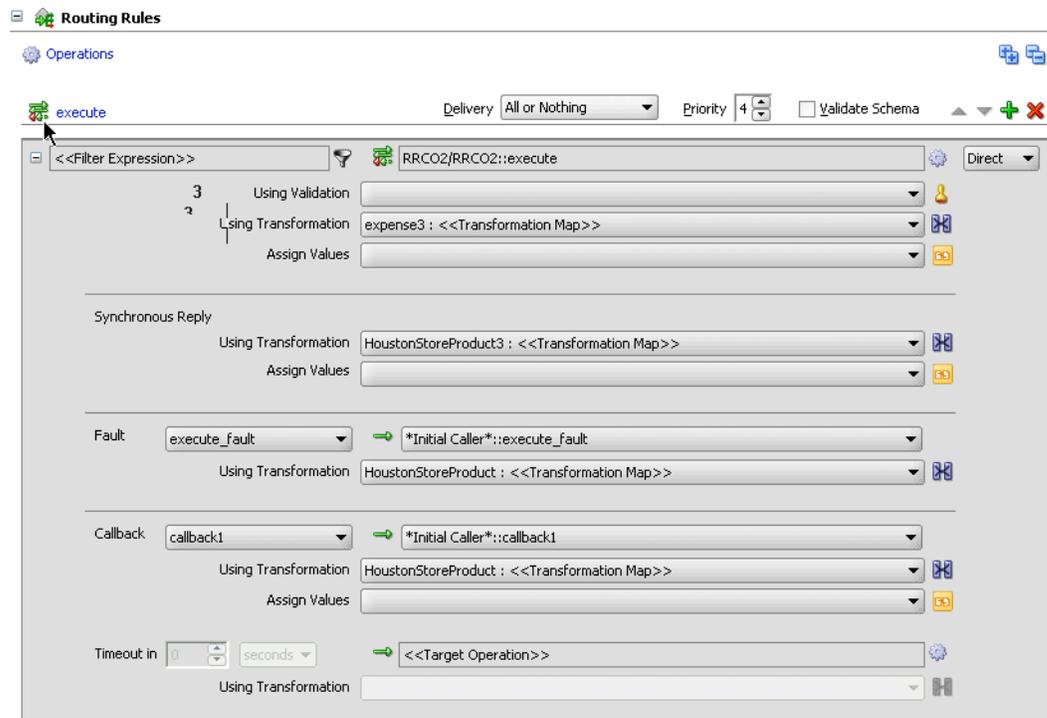
The XSLT Mapper tool is available through the Mediator editor or from the mediator icon in the **Design** tab of Oracle JDeveloper. You can either create a new transformation map or update an existing one.

To launch the XSLT Mapper from the Mediator editor and create or update a data transformation XSL file, follow these steps:

1. Open the Mediator editor.
2. Open the **Routing Rules** panel by clicking on the + icon to the left of **Routing Rules**.

The **transformation map** icon is visible in the routing rules panel.
3. Click the appropriate **transformation map** icon to the right of the **Using Transformation** field shown in [Figure 4-8](#) to open the Transformation Map window.

Figure 4–8 Routing Rules



The appropriate Transformation Map window 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 4–9 appears.

Figure 4–9 Reply Transformation Map Dialog

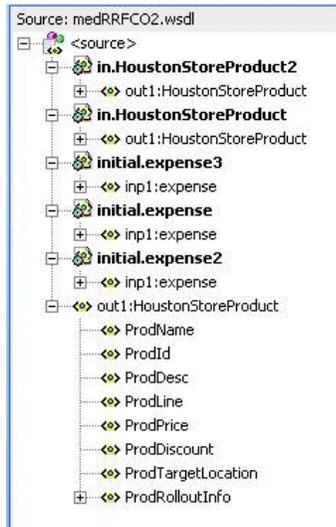


If the routing rule includes a synchronous reply or fault, the Reply Transformation Map window or Fault Transformation Map window contains the **Include Request in the Reply Payload** option. When you enable this option, you can obtain information from the request message. The request message and the 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 4–10](#) provides an example.

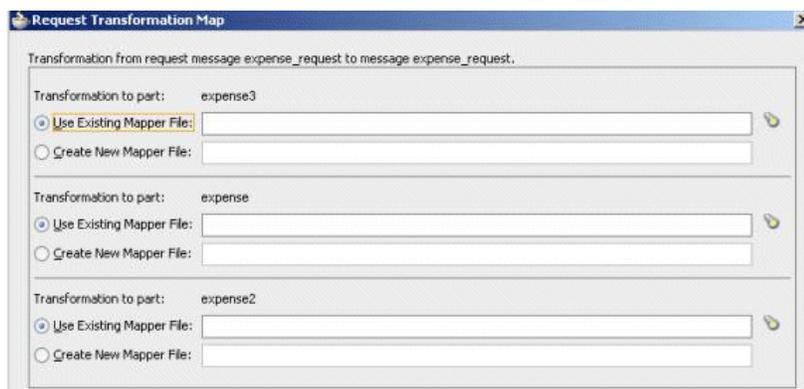
Figure 4–10 Reply Part



4. Choose one of the following options:
 - **Use Existing Mapper File** and then click the flashlight icon to browse for an existing 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 window has a separate panel for each target part. For example, here is a request where the target has three parts:

[Figure 4–11](#) provides an example.

Figure 4–11 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 4.1, "Introduction to the XSLT Mapper"](#) on page 4-1 for an overview of using the XSLT Mapper.

You can directly launch the XSLT Mapper by double-clicking on a data transformation icon in a mediator icon in the **Design** tab. If the transformation exists, the XSLT Mapper opens for you to update the transformation file. If the transformation file has not been specified yet, the Request Transformation Map window displays and enables you to create a new transformation file or select an existing transformation map file for update.

4.3 Using the XSLT Mapper

The following sections describe how to use the XSLT Mapper in Oracle BPEL Process Manager or Oracle Mediator.

- [Section 4.3.1, "How to Add Additional Sources"](#)
- [Section 4.3.2, "How to Perform a Simple Copy by Linking Nodes"](#)
- [Section 4.3.3, "How to Set Constant Values"](#)
- [Section 4.3.4, "How to Add Functions"](#)
- [Section 4.3.5, "How to Edit XPath Expressions"](#)
- [Section 4.3.6, "How to Add XSLT Constructs"](#)
- [Section 4.3.7, "How to Automatically Map Nodes"](#)
- [Section 4.3.8, "How to View Unmapped Target Nodes"](#)
- [Section 4.3.9, "How to Generate Dictionaries"](#)
- [Section 4.3.10, "How to Create Map Parameters and Variables"](#)
- [Section 4.3.11, "How to Search Source and Target Nodes"](#)
- [Section 4.3.12, "How to Control the Generation of Unmapped Target Elements"](#)
- [Section 4.3.13, "How to Ignore Elements in the XSLT Document"](#)
- [Section 4.3.14, "How to Replace a Schema in the XSLT Mapper"](#)
- [Section 4.3.15, "How to Use Type Substitution in the XSLT Mapper"](#)

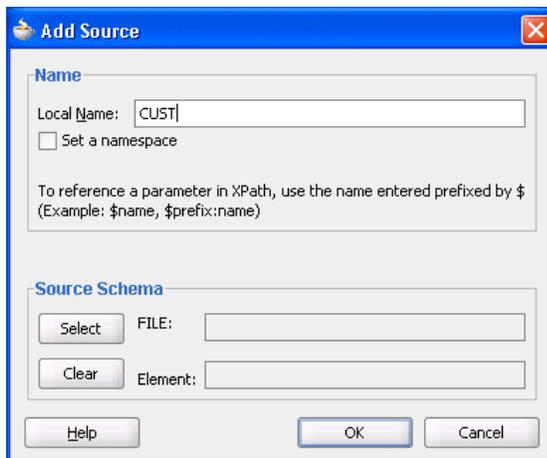
4.3.1 How to Add Additional Sources

You can add additional sources to an existing XSLT map. These sources are defined as global parameters and have schema files defining their structure.

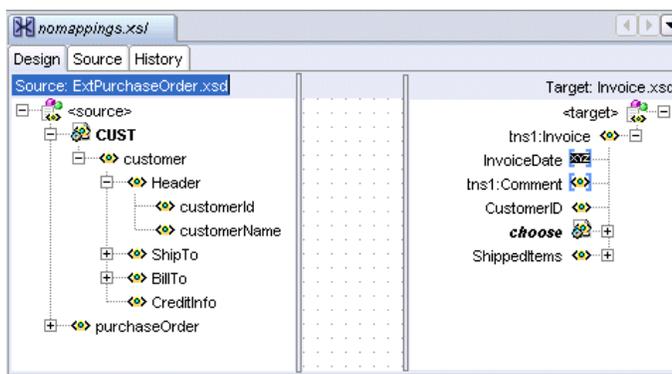
1. Right-click the source panel to display the context menu.
2. Select **Add Source**.

The Add Source window shown in [Figure 4–12](#) appears.

3. Enter a parameter name for the source (the name can also be qualified by a namespace and prefix).

Figure 4–12 Add Source Dialog

4. Click **Select** in the **Source Schema** section to select a schema for the new source.
The Type Chooser window 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 window.
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 4–13](#) to view the structure of the underlying schema.

Figure 4–13 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**).

4.3.2 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 and drop 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 4-14](#).

Figure 4-14 Linking Nodes

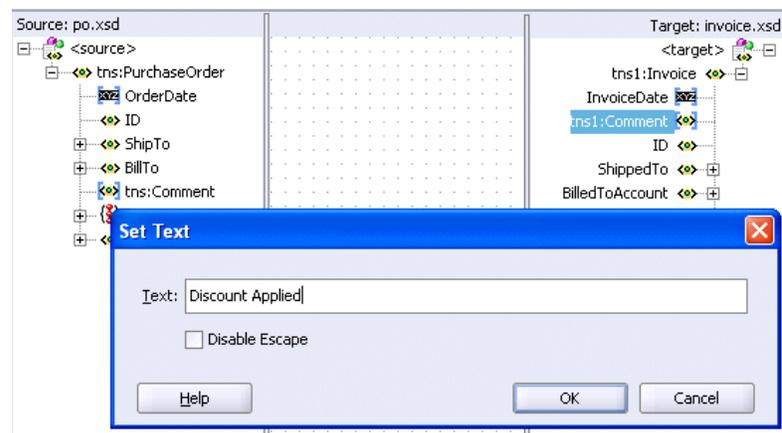


4.3.3 How to Set Constant Values

Perform the following steps to set a constant value.

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.
4. Enter text in the Set Text window (for example, **Discount Applied**, as shown in [Figure 4-15](#)).
5. Click **OK** to save the text.
A **T** icon is displayed next to the node that has text associated with it.
6. If you want to remove the text associated with the node, right click the node to invoke the Set Text window again. Delete the contents and click **OK**.

Figure 4-15 Set Text Window



For more information about the fields, see the online Help for the Set Text dialog.

4.3.4 How to Add Functions

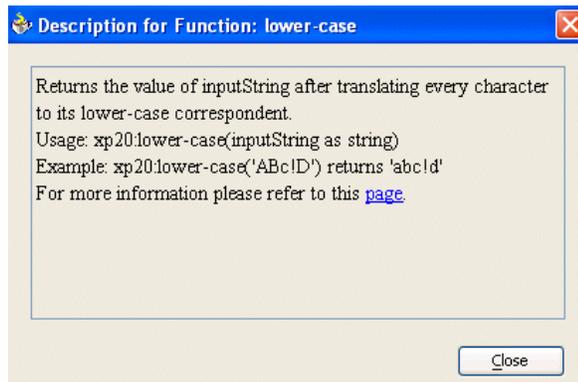
In addition to the standard XPath 1.0 functions, the XSLT Mapper provides a number of prebuilt extension functions and has the ability to support user-defined functions

and named templates. The extension functions are prefixed with **xp20** or **orcl** and mimic XPath 2.0 functions.

Perform the following steps to view function definitions and use a function:

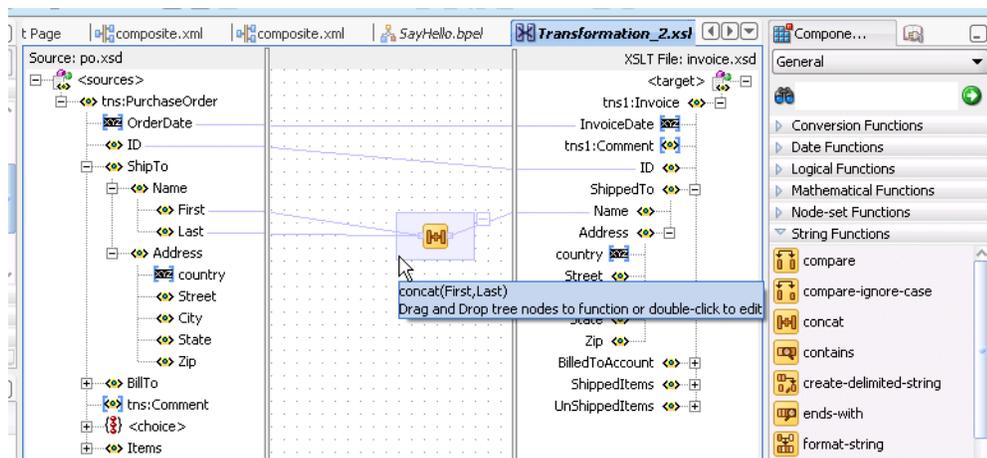
1. Select a category of functions (for example, **String Functions**) from the **Component Palette**.
2. Right-click an individual function (for example, **lower-case**).
3. Select **Help**. A dialog with a description of the function appears, as shown in [Figure 4–16](#). You can also click a link at the bottom to access this function's description at the World Wide Web Consortium at www.w3.org.

Figure 4–16 Description of Function



4. Drag a **concat** function into the mapper pane. This function enables you to connect the source parameters from the source tree to the function and the output of the function to the node on the target tree.
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 left side on the **concat** function. Also drag a thread from the **ShippedTo** name and connect it to the right side on the **concat** function, as shown in [Figure 4–17](#).

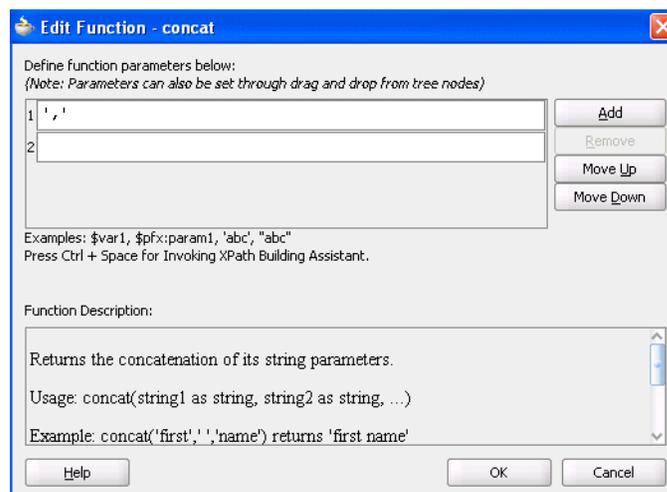
Figure 4–17 Using the Concat Function



4.3.4.1 Editing Function Parameters

To edit the parameters of the **concat** function, double-click the function icon to launch the Edit Function - concat window. This window enables you to add, remove, and reorder parameters. If you want to add a new comma parameter so that the output of the **concat** function is **Last, First**, then click **Add** to add a comma and reorder the parameters to get this output.

Figure 4–18 *Editing Function Parameters*



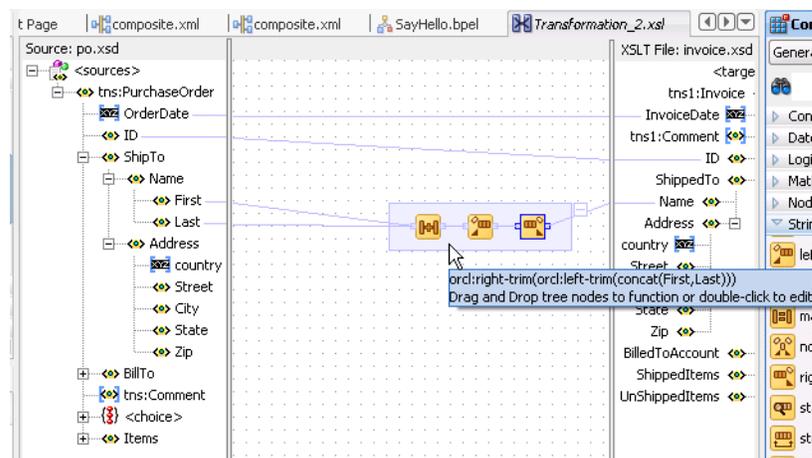
For more information about how to add, remove, and reorder function parameters, see the online Help for the Edit Function dialog.

4.3.4.2 Chaining Functions

Complex expressions can be built by chaining functions. To remove all leading and trailing spaces from the output of the above **concat** function, use the left-trim and right-trim functions and chain them as shown in the [Figure 4–19](#).

The chaining function can also be defined by dragging and dropping the function to a connecting link.

Figure 4–19 *Chaining Functions*



4.3.4.3 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 4.3.6.6, "Including External Templates with xsl:include."](#)

4.3.4.4 Importing User-Defined Functions

Follow these steps to create and use your own functions. All sample documents mentioned in these steps are found in the following directory:

`SOA_Oracle_HOME\bpel\samples\demos\XSLMapper\ExtensionFunctions`

1. Create an XML extension function configuration file. This file defines the functions and their parameters. For example, the XML code shown in [Example 4–5](#) is from the `SampleExtensionFunctions.xml` file.

Example 4–5 XML Extension Function Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<extension-functions>
  <functions
    xmlns:sample="http://www.oracle.com/XSL/Transform/java/oracle.sample.
      SampleExtensionFunctions">
    <!-- converts a value in pounds to kilograms -->
    <function name="sample:toKilograms" as="number">
      <param name="pounds" as="number"/>
    </function>

    <!-- returns a new string resulting from replacing all
      occurrences of oldChar in this string with newChar -->
    <function name="sample:replaceChar" as="string">
      <param name="inputString" as="string"/>
      <param name="oldChar" as="string"/>
      <param name="newChar" as="string"/>
    </function>

  </functions>
</extension-functions>
```

This file defines the following sample functions:

- `replaceChar(inputString, oldChar, newChar)` returns a new string resulting from replacing all occurrences of `oldChar` with `newChar`. For example, `replaceChar('_*_','*','_')` is `'___'`.
- `toKilograms(pounds)` takes a number in pounds and converts it to kilos. For example, `toKilograms(number('10'))` is `4.5359237`. You must convert the `toKilograms()` input value to a number before calling it.

Observe the following rules in the creation of the XML file:

- Your 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`.
- For extension functions to work with the Oracle XSLT processor, the function namespace must start with `http://www.oracle.com/XSL/Transform/java/`.
- 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 following types are accepted (the `as` attribute):
 - string
 - boolean
 - number
 - node-set
 - tree

2. Code and build your functions.

The XSL Mapper extension functions are coded differently than the Oracle BPEL Process Manager extension functions. Two examples are provided in the `SampleExtensionFunctions.java` file. [Example 4–6](#) provides an example.

Example 4–6 XSL Mapper Extension Functions

```
// SampleExtensionFunctions.java
package oracle.sample;
/*
This is a sample XSL 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));
    }
}
```

3. Copy your JAR file to the `JDEV_HOME\jdev\lib\patches` directory. This is required for the implementation of your functions to be part of the Oracle

JDeveloper class path. For example, to use the sample functions, copy `SampleExtensionFunctions.jar` to the `patches` directory.

- Go to **Tools > Preferences > XSL Map** in Oracle JDeveloper and add your extension function configuration XML file name to the **User Defined Extension Functions Config File** field. (For example, to use the sample functions, add `SampleExtensionFunctions.xml`).

- Restart Oracle JDeveloper.

New functions appear in the **Component Palette** under the **User Defined** page in the **User Defined Extension Functions** group. If you are using the sample functions, `replaceChar()` and `toKilograms()` appear under the **User Defined** page in the **User Defined Extension Functions** group when an XSL map is open.

- If the XSL generated by the XSLT Mapper is to be used by Oracle Application Server, add the JAR file to the Oracle Application Server class path. For OC4J, copy the JAR file to the following location:

```
$INSTANCE_HOME/OC4JComponent/oc4j_soa/applib
```

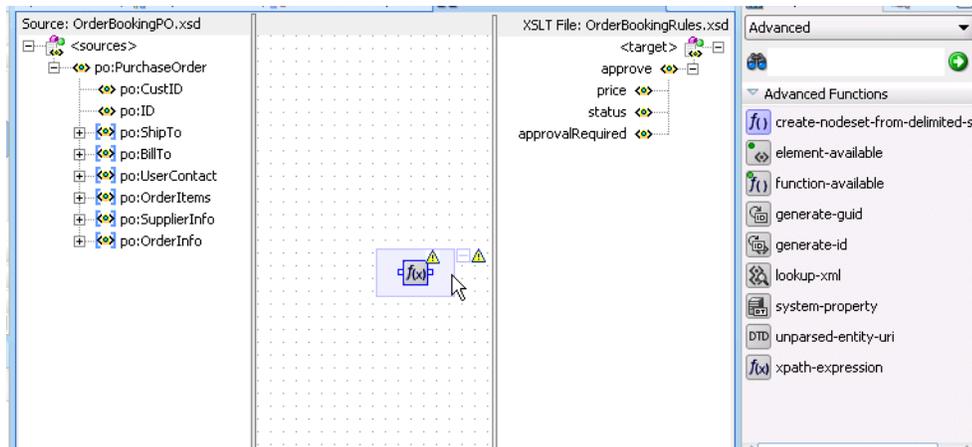
Note: This path is version dependent.

- For 11.1.3, copy the JAR file to `OC4J_HOME\j2ee\home\applib`.
 - For 10.1.2, copy the JAR file to `ORACLE_HOME/j2ee/OC4J_BPEL/applib`.
-

4.3.5 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 and drop `xpath-expression` from the list into the transformation window. This is shown in [Figure 4–20](#).

Figure 4–20 Editing XPath Expressions



When you double-click the icon, the Edit XPath Expression window appears, as shown in [Figure 4–21](#). You can press the **Ctrl** key and then the **spacebar** to invoke the XPath Building Assistant.

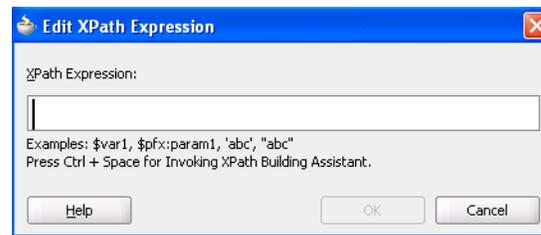
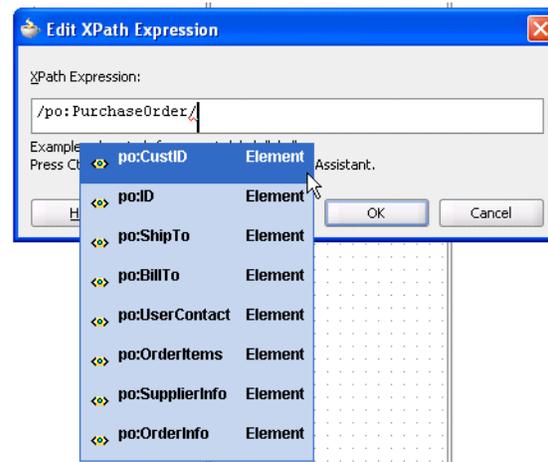
Figure 4–21 Edit XPath Expression Window

Figure 4–22 shows the XPath Building Assistant.

Figure 4–22 The XPath Building Assistant

For more information about using the XPath Building Assistant, see the online Help for the Edit XPath Expression dialog.

4.3.6 How to Add XSLT Constructs

While mapping complex schemas, it is sometimes essential to conditionally map a source node to a target or map a node-set of elements in the source to a node-set of elements in the target. The XSLT Mapper provides various XSLT constructs.

There are two ways to add XSLT constructs such as **for-each**, **if**, or **choose** to the target XSLT tree:

- Through drag and drop
 1. Select the **General** page of the **Component Palette** and open the **XSLT Constructs** group.
 2. Drag and drop 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.
- Through the context menu on the target tree
 1. Right-click the element in the target tree *before* which you want to insert an XSLT construct. A context menu is displayed.
 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. In previous sections, the following methods for mapping elements and attributes were described:

- Creating a simple copy by linking nodes
- Adding functions
- Adding XPath expressions

Each of these methods creates 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.

The following sections describe specific steps for inserting each supported XSLT construct.

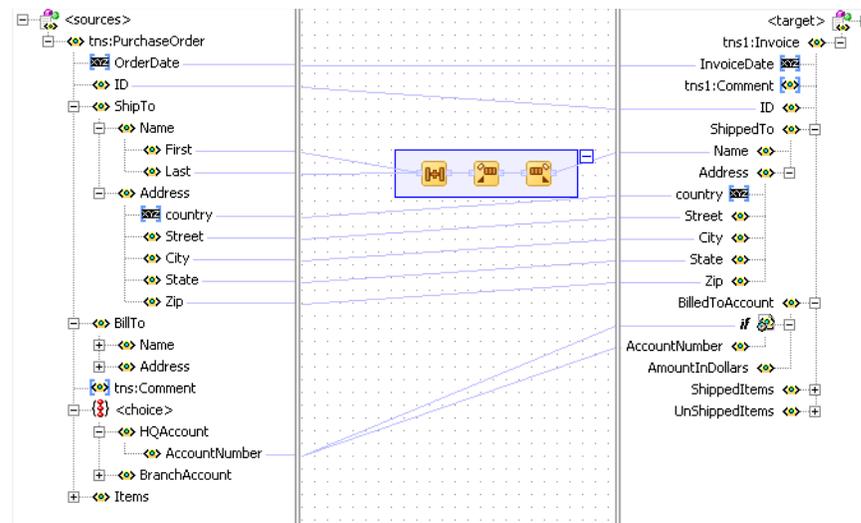
- [Section 4.3.6.1, "Using Conditional Processing with `xsl:if`"](#)
- [Section 4.3.6.2, "Using Conditional Processing with `xsl:choose`"](#)
- [Section 4.3.6.3, "Creating Loops with `xsl:for-each`"](#)
- [Section 4.3.6.5, "Copying Nodes with `xsl:copy-of`"](#)
- [Section 4.3.6.6, "Including External Templates with `xsl:include`"](#)

4.3.6.1 Using Conditional Processing with `xsl:if`

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 do this:

1. Select **Invoice/BilledToAccount/AccountNumber** in the target tree and right-click to bring up 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 4–23](#) shows the results.

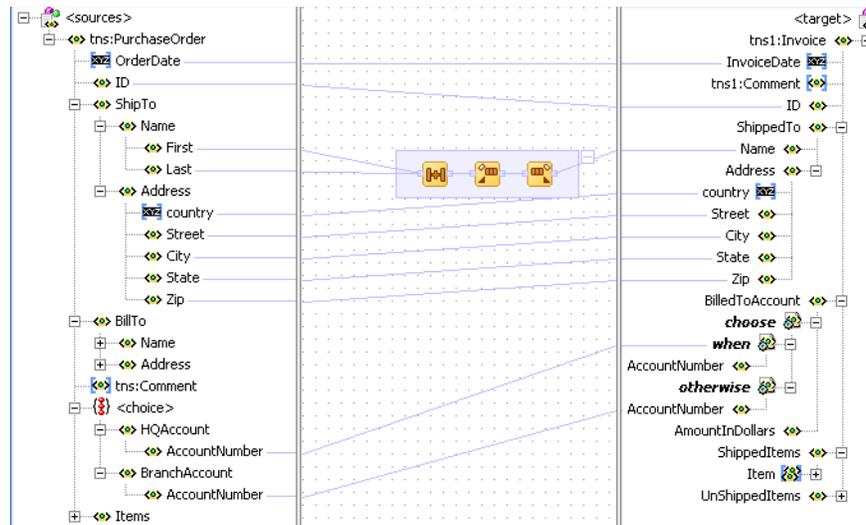
Figure 4–23 Conditional Processing with *xsl:if*

4.3.6.2 Using Conditional Processing with *xsl:choose*

You can copy **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/AccountNumber**, if it exists. Otherwise, copy **PurchaseOrder/BranchAccount** to **Invoice/BilledToAccount/AccountNumber** as follows:

1. Select **Invoice/BilledToAccount/AccountNumber** in the target tree and right-click to bring up the context sensitive menu.
2. Select **Add XSL Node > choose** and connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/choose/when** to define the condition.
3. Connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/choose/when/AccountNumber**.
4. Select **XSL Add Node > choose** in the target tree and right-click to bring up the context sensitive menu.
5. Select **Add XSL node > otherwise** from the menu.
6. Connect **PurchaseOrder/BranchAccount/AccountNumber** to **Invoice/BilledToAccount/choose/otherwise/AccountNumber**.

Figure 4–24 shows the results.

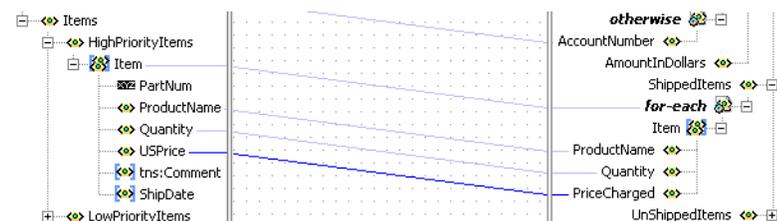
Figure 4–24 Conditional Processing with *xsl:choose*

4.3.6.3 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** as follows:

1. Select **Invoice/ShippedItems/Item** in the target tree and right-click to bring up 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 4–25 shows the results.

Figure 4–25 Creating Loops with *xsl:for-each*

Note: Executing an auto map automatically inserts *xsl:for-each*. To see the auto map in use, drag and drop **PurchaseOrder/Items/LowPriorityItems** to **Invoice/UnShippedItems**; *for-each* is automatically created.

4.3.6.4 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`, as follows:

1. Select `for-each` under `Invoice/ShippedItems`.
2. Right-click and select **Add XSL Node > Clone 'for-each'**.
This inserts a copy of the `for-each` node below 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`.

4.3.6.5 Copying Nodes with `xsl:copy-of`

You may need to use the XSLT `copy-of` command 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.

Insert a `copy-of` command as follows:

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 window 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.

4.3.6 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.

Insert an include statement as follows:

1. Select and then right-click the root node of the target tree.
2. Select **Add Include File** from the menu.

A window prompts you for the include file name.

3. Select the file and click **OK**.

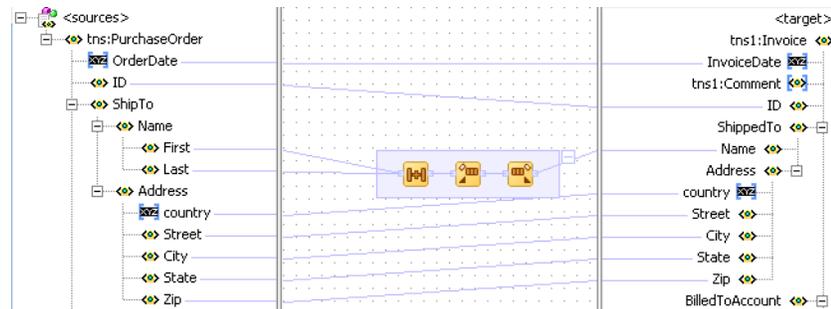
The file is copied to the same project directory as the existing map file. A relative path name is created for it and the include statement instruction is inserted in the target tree.

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.

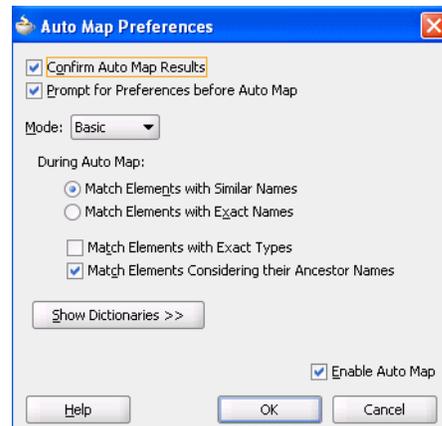
4.3.7 How to Automatically Map Nodes

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 4-26](#).

Figure 4-26 Auto Mapping



The behavior of the auto map can be tuned by altering the settings in Oracle JDeveloper preferences or by right-clicking the transformation window and selecting **Auto Map Preferences**. This displays the window shown in [Figure 4-27](#).

Figure 4–27 Auto Map Preferences

This window enables you to customize your auto mapping as follows:

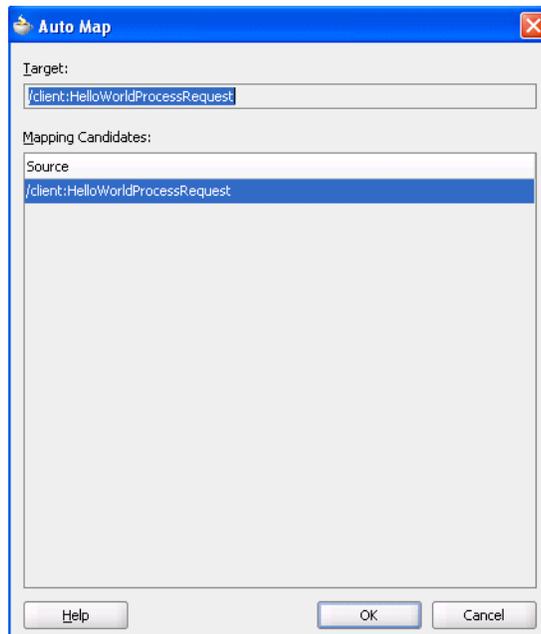
- Invoke the automatic mapping feature, which attempts to automatically link all relevant nodes under the selected source and target. When disabled, you must individually map relevant nodes.
- Display and review all potential source-to-target mappings detected by the XSLT Mapper, and then confirm to 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 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 window.

Follow these instructions to see potential source mapping candidates for a target node.

1. Right-click the target node and select **Show Matches**.
2. Click **OK** in the Auto Map Preferences window.

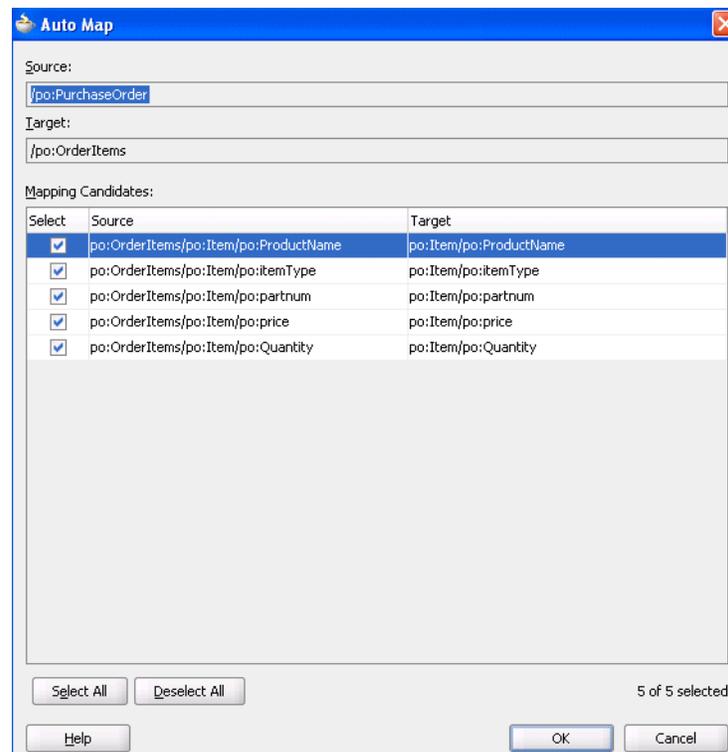
The Auto Map window appears, as shown in [Figure 4–28](#).

Figure 4–28 Auto Mapping Candidates

For more information on the fields, see the online Help for the Auto Map dialog.

4.3.7.1 Using Auto Mapping with Confirmation

When the **Confirm Auto Map Results** check box shown in [Figure 4–27](#) is selected, a confirmation window appears. If matches are found, the potential source-to-target mappings detected by the XSLT Mapper are displayed, as shown in [Figure 4–29](#). The window enables you to filter one or more mappings.

Figure 4–29 Auto Map with Confirmation

For more information about the fields, see the online Help for the Auto Map dialog.

4.3.8 How to View Unmapped Target Nodes

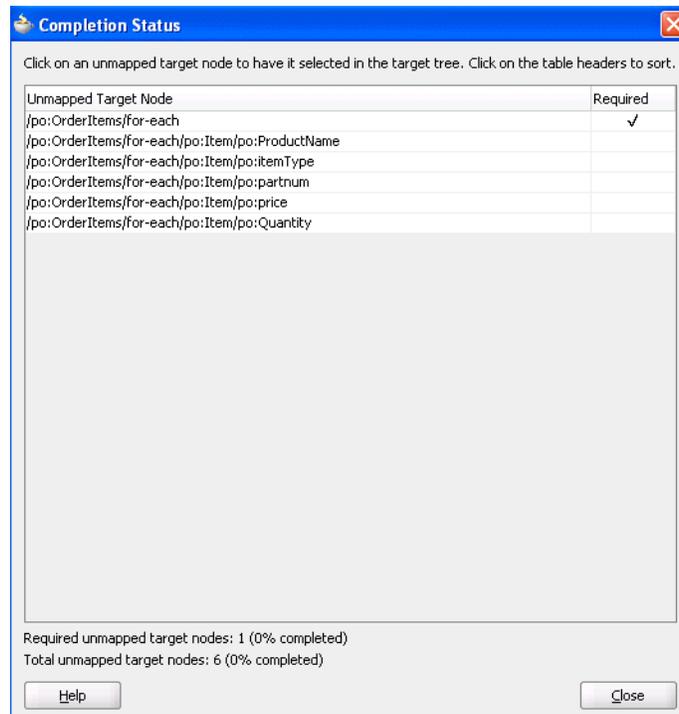
You can view a list of target nodes that are currently unmapped to source nodes.

1. Right click in the mapper pane and select **Completion Status**.

This window provides statistics at the bottom about the number of unmapped target nodes. This window enables you to identify and correct any unmapped nodes before you test your transformation mapping logic on the Test XSL Map window.

2. Select a target node in the list. The node is highlighted. A check mark indicates that the target node is required to be mapped. If not required, the check box is empty.

Figure 4–30 provides an example of the Completion Status window.

Figure 4–30 Completion Status

4.3.9 How to Generate Dictionaries

A dictionary is an XML file that captures the synonyms for mappings. You can reuse these mapping definitions. For example, you may want to map a purchase order to a purchase order acknowledgment, then reuse most of the mapping definitions later.

1. Right-click the mapper pane and select **Generate Dictionary**. This prompts you for the dictionary name and the directory in which to place the dictionary. The XSLT Mapper uses the rules defined in the dictionary when attempting to automatically map source and target elements.
2. Build all the mapping logic for the purchase order and purchase order acknowledgment.
3. Generate a dictionary for the created map.
4. Create a new map using a *different* purchase order and purchase order acknowledgment.
5. Load the previously created dictionary by selecting **Preferences > XSL Maps > Auto Map** in the **Tools** main menu of Oracle JDeveloper.
6. Perform an automatic mapping from the purchase order to the purchase order acknowledgment.

4.3.10 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. Where they are defined in the target tree determines if they are global or local.
 - Global variables are defined immediately below the **<target>** node and immediately above the actual target schema (for example, **POAcknowledge**). Right-click on the **<target>** node to create a global variable.
 - Local variables are defined on a specific node below 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.

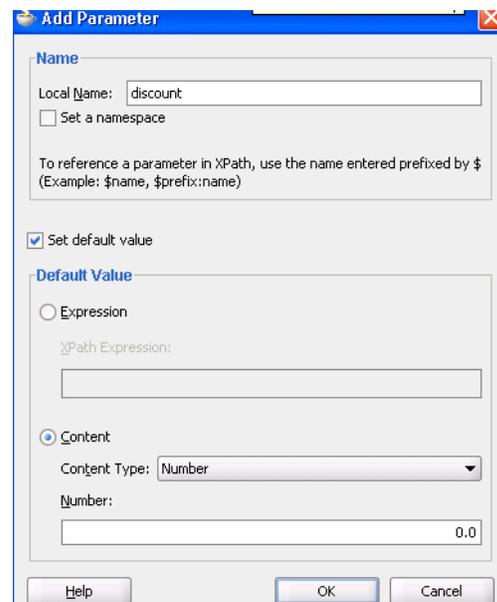
4.3.10.1 Creating a Map Parameter

1. Right-click the source tree root and select **Add Parameter**.

The Add Parameter window shown in [Figure 4–31](#) 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 4–31 Add Parameter Dialog



3. Click **OK**.

4.3.10.2 Creating a Map Variable

1. Right-click the target tree root or any node in the target tree and select **Add Variable**.

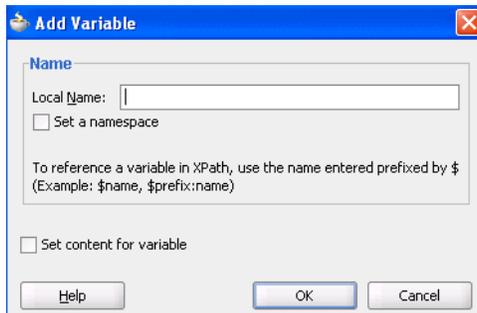
The Add Variable window shown in [Figure 4–32](#) appears.

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 window is a name for the variable. If you want to set content for the variable, you must do it through this

window. Content is handled differently from the XSLT select attribute on the variable.

Figure 4–32 Add Variable Dialog



3. Click **OK**.

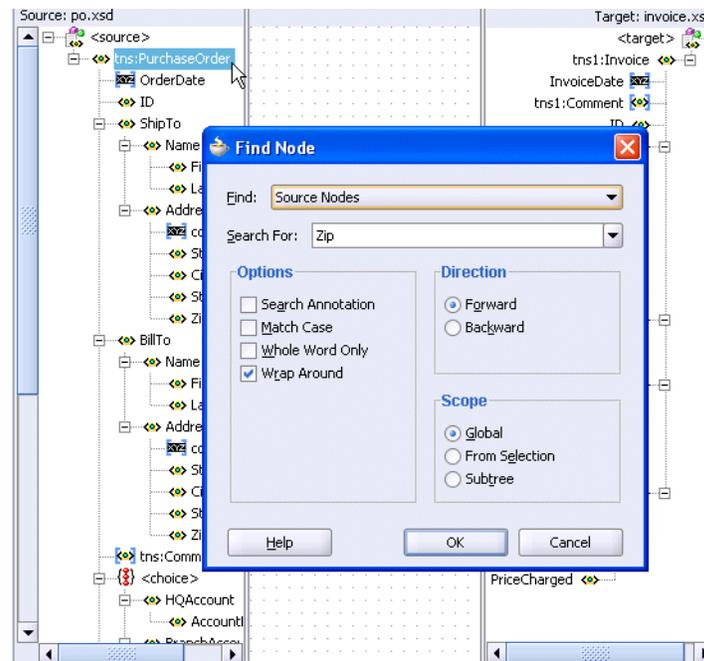
The variable is added to the target tree at the position selected.

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.

4.3.11 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**.

1. Right-click a source or target node. [Figure 4–33](#) shows the Find Node dialog.
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 4–33 Find Node Dialog

The first match found is highlighted, and the Find window 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.

4.3.12 How to Control the Generation of Unmapped Target Elements

There are three 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.

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.

4.3.13 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 is unable to 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.

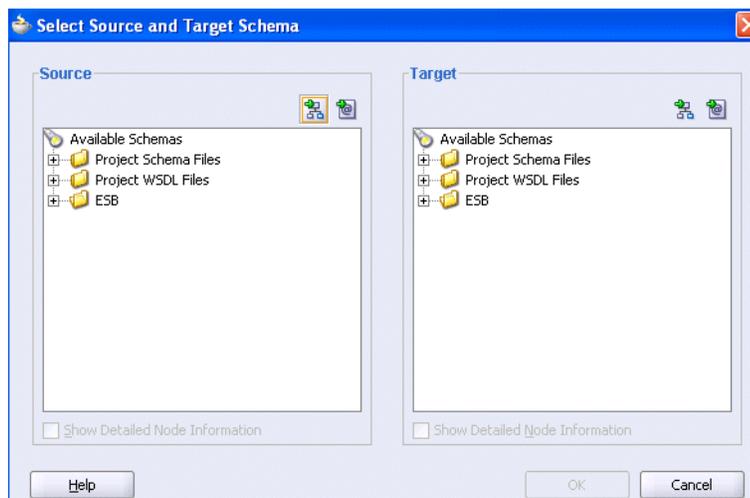
4.3.14 How to Replace a Schema in the XSLT Mapper

You can replace the map source schema and map target schema that currently display in the XSLT Mapper.

1. Right click in either the source or target panel and select **Replace Schema**.

This opens the Select Source and Target Schema window shown in [Figure 4–34](#), which enables you to select the new source or target schema to use.

Figure 4–34 Replacing a Schema



4.3.15 How to Use Type Substitution in the XSLT Mapper

The `xsi:type` attribute in the XML schema instance namespace enables you to substitute a defined global type for a type used in the target schema if the substituted type is derived from the type used in a schema element. For example, assume your XSD file contains the global type definitions shown in [Example 4–7](#):

Example 4–7 XSD File with Global Type Definitions

```
<complexType name="Address">
  <sequence>
    <element name="name" type="string"/>
    <element name="street" type="string"/>
    <element name="city" type="string"/>
  </sequence>
```

```

</complexType>

<complexType name="USAddress">
  <complexContent>
    <extension base="ipo:Address">
      <sequence>
        <element name="state" type="string"/>
        <element name="zip" type="positiveInteger"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="UKAddress">
  <complexContent>
    <extension base="ipo:Address">
      <sequence>
        <element name="postcode" type="string"/>
      </sequence>
      <attribute name="exportCode" type="positiveInteger" fixed="1"/>
    </extension>
  </complexContent>
</complexType>

```

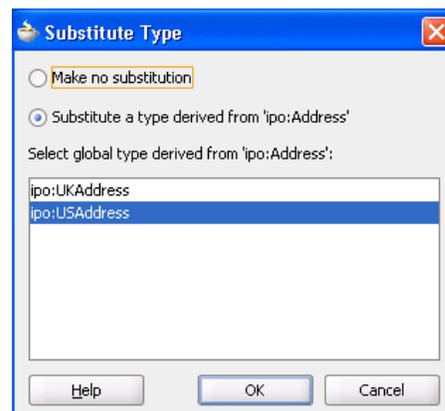
If a schema element is defined to be of type `Address`, you may substitute `USAddress` or `UKAddress` for that schema element by adding an `xsi:type` attribute to the resulting target XML.

You must currently add the specific mappings to the substituted type in source view, but some features have been added to design view to aid in the process.

To use type substitution in the target schema:

1. Right-click the element in the target schema for which substitution applies.
2. Select **Substitute Type** from the context menu. The dialog shown in [Figure 4–35](#) appears.

Figure 4–35 Substitute Type

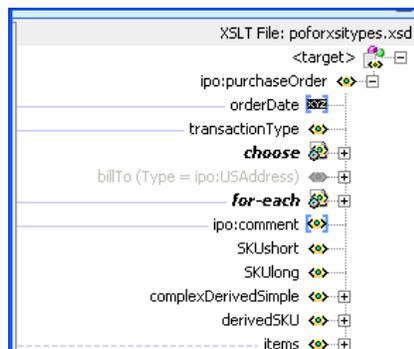


3. Select **Substitute a type derived from *type*** to display a list of possible global types for substitution. To remove a substitution, select **Make no Substitution**. If you select **Make no Substitution**, note the following details:
 - If no type was substituted for the base type and this is the first time you are seeing this window, the behavior is the same as clicking **Cancel**.

- If a type was substituted earlier and you are returning to this window, a warning message appears indicating that if you continue, your mappings are removed. Click **Yes** to continue. All mappings under the node are removed and you can continue from there.
4. Click **OK**.

The originally-selected element becomes uneditable in design view and appears grayed out in the tree. The type that is substituted is indicated. For example, the tree shown in [Figure 4-36](#) indicates that the **billTo** element, which was defined as the **Address** type, has now been substituted with the **USAddress** type.

Figure 4-36 *billTo Element Substituted with USAddress Type*



5. Switch to source view and note that the code shown in [Example 4-8](#) appears for the **billTo** element:

Example 4-8 *billTo Element Code*

```
<billTo>
  <xsl:attribute name="xsi:type">
    <xsl:value-of select="'ipo:USAddress'"/>
  </xsl:attribute>
  <!--Add custom mapping below-->
</billTo>
```

6. Add your own mapping code below the comment. [Example 4-9](#) provides an example.

Example 4-9 *Mapping Code*

```
<billTo>
  <xsl:attribute name="xsi:type">
    <xsl:value-of select="'ipo:USAddress'"/>
  </xsl:attribute>
  <!--Add custom mapping below-->
  <name>
    <xsl:value-of select="/ipo:purchaseOrder/billTo/name"/>
  </name>
  <street>
    <xsl:value-of select="/ipo:purchaseOrder/billTo/street"/>
  </street>
  <city>
    <xsl:value-of select="/ipo:purchaseOrder/billTo/city"/>
  </city>
  <state>
    <xsl:value-of select="/ipo:purchaseOrder/billTo/state"/>
  </state>
```

```

</state>
<zip>
  <xsl:value-of select="/ipo:purchaseOrder/billTo/zip" />
</zip>
</billTo>

```

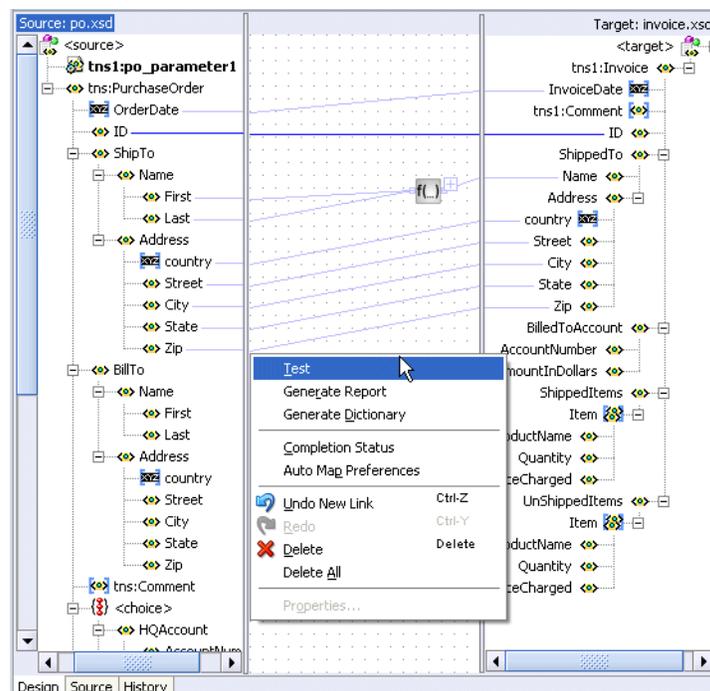
7. Switch back to design view. Your code is preserved, and not removed. However, the code is only editable in source view.

Note that no features are provided to perform type substitution on the source schema. However, it is possible to create an XPath statement referencing nodes in the source tree that are not defined in the XSD. This XPath should not create an error, but should be kept as a constant XPath expression. An XPath of this type can be created using the `xpath-expression` function under the **Advanced** page/**Advanced Function** group of the **Component Palette**.

4.4 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 4-37](#).

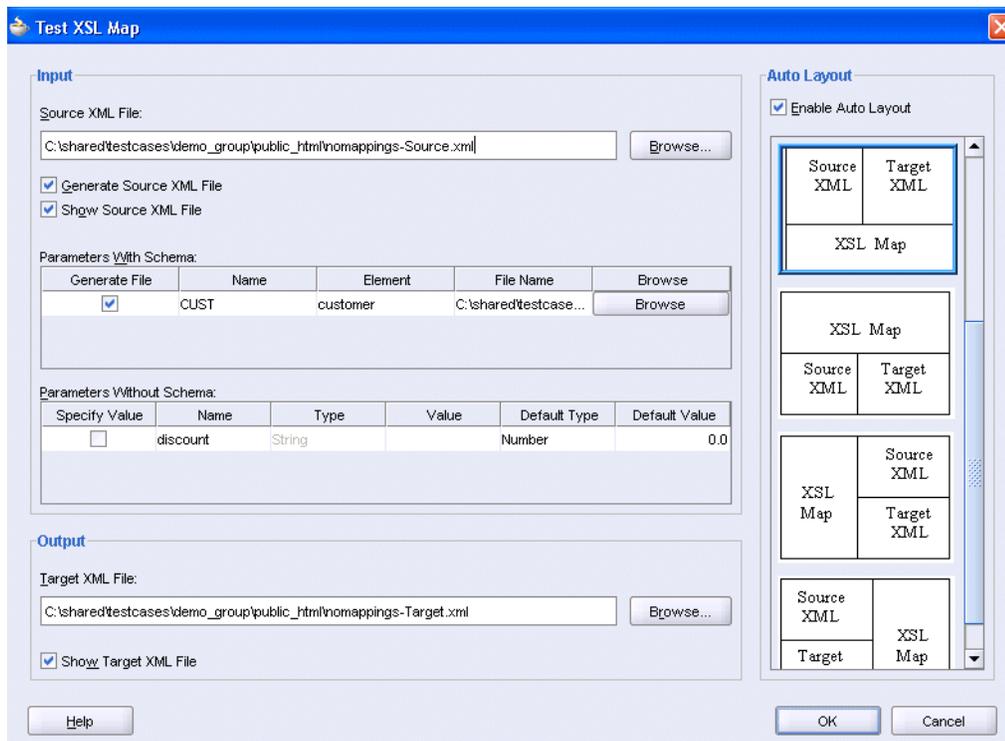
Figure 4-37 Invoking the Test Window



4.4.1 How to Test the Transformation Mapping Logic

The Test XSL Map window shown in [Figure 4-38](#) 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 4–38 Test XSL Map Window



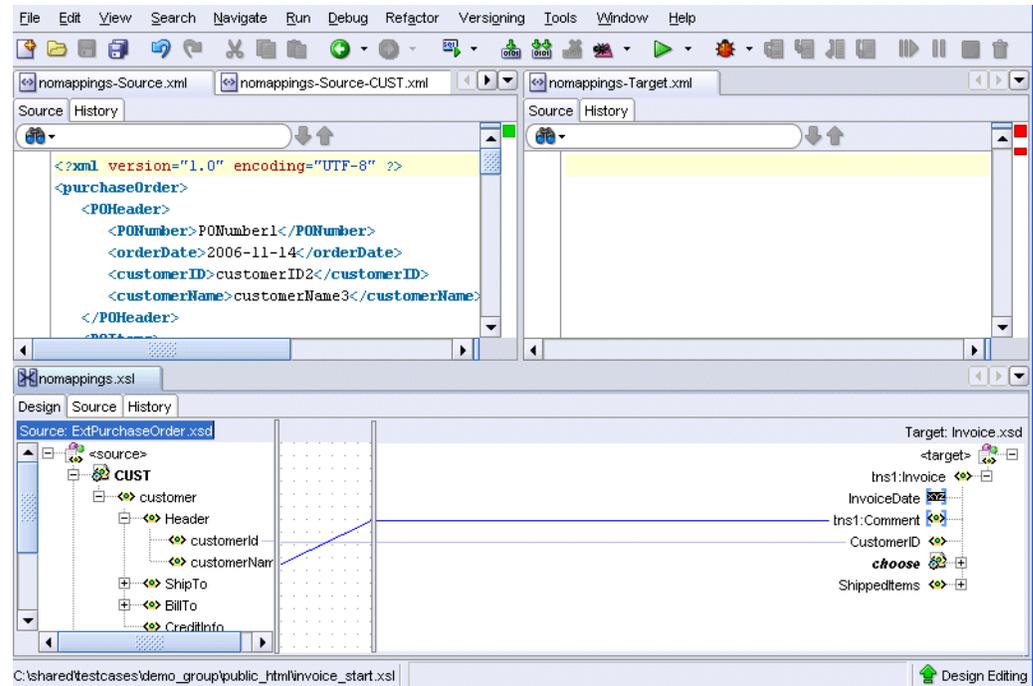
- Choose to allow a sample source XML file to be generated for testing or click **Browse** to specify a different source XML file in the **Source XML File** field.
When you click **OK**, the source XML file is validated. If validation passes, transformation occurs, and the target XML file is created.
If validation fails, no transformation occurs and a message displays on-screen.
- Select the **Generate Source XML File** check box to create a sample XML file based on the map source XSD schema.
- Select the **Show Source XML File** check box 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.
 - 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** check box if you want to generate a file.
 - If the **Parameters Without Schema** table appears, you can specify a value by selecting the **Specify Value** check box and making appropriate edits to the **Type** and **Value** columns.
- Enter a file name in the **Target XML File** field or browse for a file name in which to store the resulting XML document from the transformation.
- Select the **Show Target XML File** check box to display the target XML file for the test. The target XML file displays in an Oracle JDeveloper XML editor.
- 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 4-39](#) 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 4-39 Test Results



4.4.2 How to Generate Reports

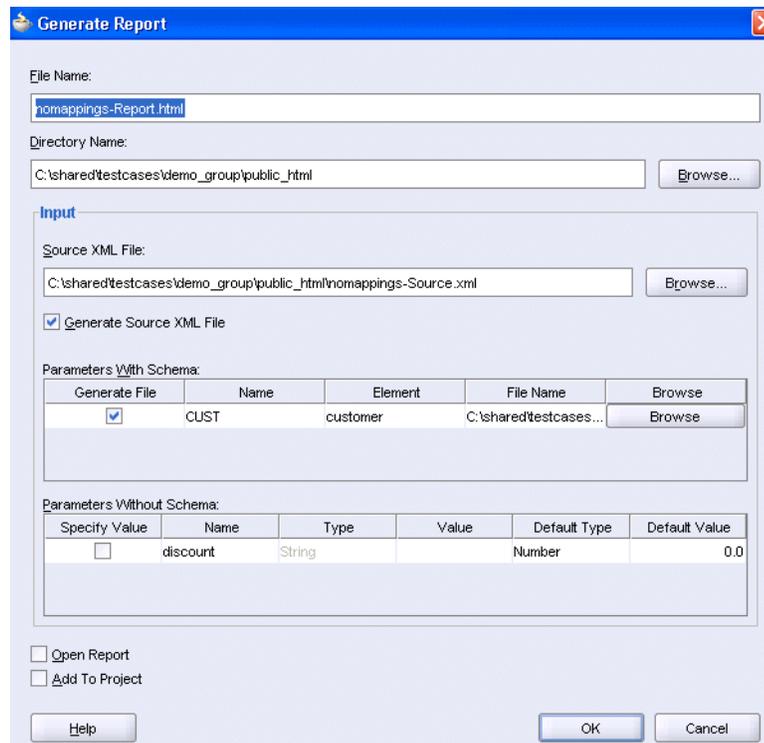
You can generate an HTML report with the following information:

- 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. Right-click the transformation window and select **Generate Report**.

The Generate Report window appears in the transformation window, as shown in [Figure 4-40](#). Note that if the map has defined parameters, the appropriate parameter tables appear.

Figure 4–40 The Generate Report Window

For more information about the fields, see the online Help for the Generate Report dialog.

4.4.2.1 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:

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 window before generating the report.

4.4.3 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 window.

- 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.

4.5 Use Case for Transformation

Transformation use is demonstrated in several use cases.

Note: The `XSLMapper` tutorial demonstrated in this chapter is not included in the `soa-samples.zip` file for beta 2.

Getting Started with Oracle Mediator

This chapter provides an overview of Oracle Mediator and describes how to create an Oracle Mediator service component by using a service invocation or an event subscription as entry point.

This chapter includes the following topics:

- [Introduction to Oracle Mediator](#)
- [Overview of Mediator Designer Environment](#)
- [Creating a Mediator Component](#)
- [Generating a WSDL File](#)
- [Specifying Operation or Event Subscription Properties](#)
- [Modifying a Mediator component](#)
- [Deleting a Mediator component](#)

5.1 Introduction to Oracle Mediator

Oracle Mediator provides a lightweight framework to mediate between various producers and consumers of services and events. In most business environments, customer data resides in disparate sources including business partners, legacy applications, enterprise applications, databases, and custom applications. The challenge of integrating this data can be met by using Oracle Mediator to deliver appropriate real-time data access to all applications that update or have a common interest in the same data. For example, an Oracle Mediator service component (mediator component) 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 services 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 invokes another service or raises another business event. You can use a mediator component to handle returned responses, callbacks, faults, and timeouts. In addition, you can also implement a variety of integration patterns such as service virtualization, publish and subscribe, fan-in, and fan-out and various synchronous and asynchronous request response patterns.

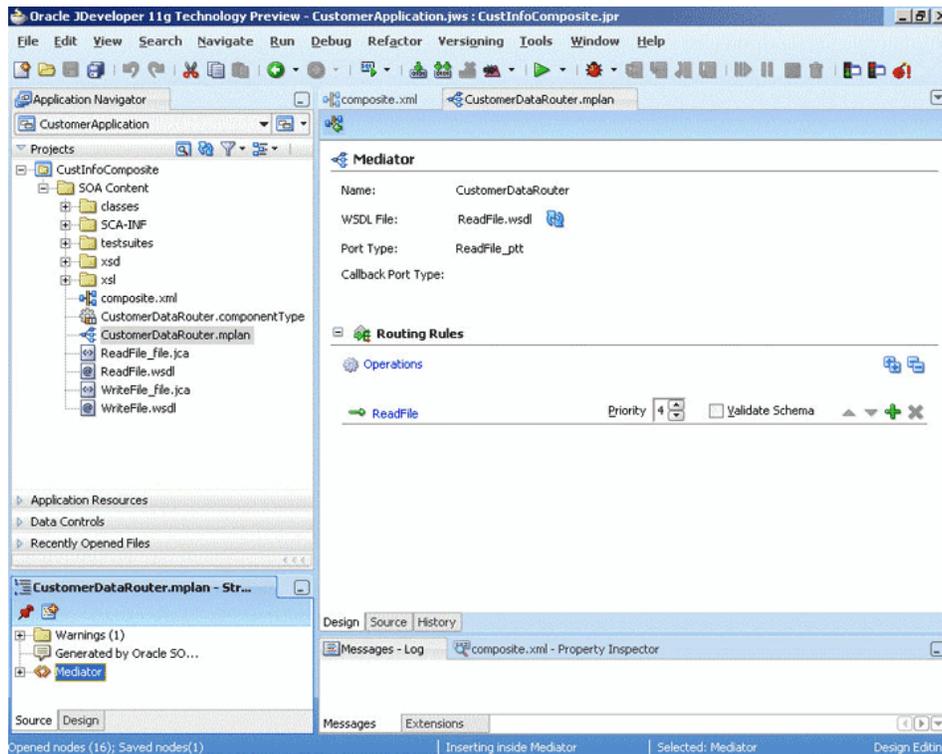
5.2 Overview of Mediator Designer Environment

You can create a Mediator component 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 component in SOA Composite Editor.

See Also: [Chapter 2, "Introduction to the SOA Composite Editor"](#) for information about SOA Composite Editor.

[Figure 5–1](#) shows the Mediator Editor along with Application Navigator, Structure, and Messages windows.

Figure 5–1 Mediator Design Window

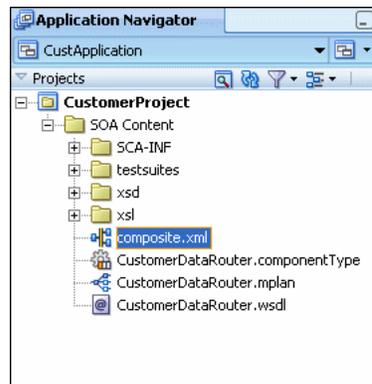


Each section of the view shown in [Figure 5–1](#) enables you to perform specific design and deployment tasks. The following list describes these sections and their functionality:

- Application Navigator

The Application Navigator shown in the upper left part of [Figure 5–1](#) displays the Mediator component files. [Figure 5–2](#) shows the files that appear under the SOA Content folder when you create a Mediator component in a SOA Composite application.

Figure 5–2 Mediator Files in Application Navigator



As shown in [Figure 5–2](#), a SOA Composite application consists of the following Mediator files:

- `Composite.xml`: The file that describes the entire SOA composite application.

See Also: [Chapter 2, "Introduction to the SOA Composite Editor"](#) for specific details about this file.

- `CustomerDataRouter.componentType`: The `.componentType` file describes the services and references for a service component.
- `CustomerDataRouter.mplan`: The `.mplan` file contains Mediator metadata.
- `CustomerDataRouter.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 5–1](#), provides a visual view of the Mediator component that you have created. This view is displayed when you perform one of the following actions:

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

- **Source View**

The Source View enables you to view the source code of a Mediator component. Click Source at the bottom of the Design window shown in [Figure 5–1](#) 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 component source code:

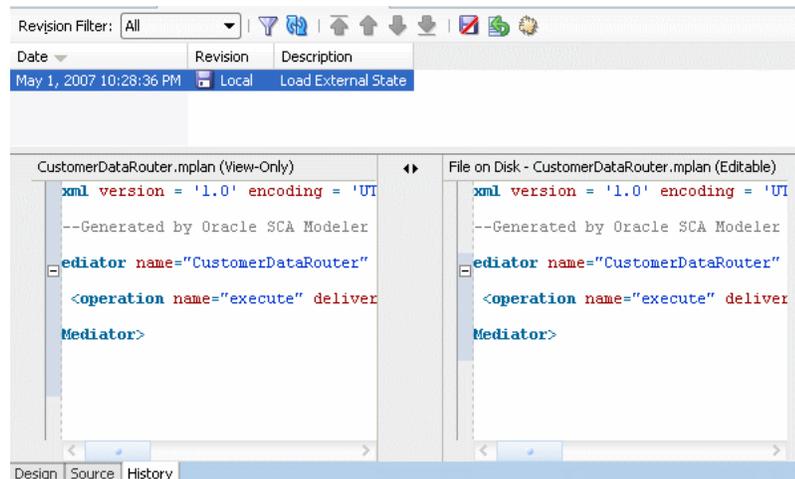
```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!--Generated by Oracle SCA Modeler version 1.0 at [4/16/07 10:05 PM].-->
<Mediator name="CustomerDataRouter"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.oracle.com/sca/1.0/mediator"/>
```

- **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 5-1](#) to open the History window. [Figure 5-3](#) shows the History view for a Mediator component file.

Figure 5-3 History Window



- **Property Inspector**
The Property Inspector shown at the bottom of [Figure 5-1](#) enables you to view details about Mediator component properties.
- **Structure Window**
The Structure Window shown in the lower left part of [Figure 5-1](#) provides a structural view of the data of a Mediator component.
- **Log Window**
The Log Window displays messages about the status of validation and compilation.

5.3 Creating a Mediator Component

You can create a Mediator component in a SOA Composite application of Oracle JDeveloper by using one of the following methods:

- By dragging and dropping a Mediator component component from the Component Palette.
- By selecting Composite with Mediator component in the Create SOA Composite dialog or Create SOA Project dialog.
- By selecting **Service Components** from Categories and **Mediator** from Items in the New Gallery dialog.

Each method opens the Create Mediator dialog where you specify the name of the Mediator component and select a template. A template provides a basic set of default files with which you can begin designing your Mediator.

5.3.1 Creating a Medator Component Without Interface Definition

You can create an empty Mediator component with no interface definition. This provides you the flexibility to create the SCA components in the order you want. For

example, you can create a Mediator component first and then create a service or an event that will initiate the Mediator component.

5.3.1.1 How to Create a Mediator With No Interface Definition

You can create an empty Mediator component 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** and click **OK**.

5.3.2 Creating a Mediator Based on a WSDL File

You can create a Mediator component based on an existing WSDL file. A WSDL file describes the interface of a Mediator component such as schemas and operations.

5.3.2.1 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 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**.
4. Deselect the Create Composite Service with SOAP Bindings option if you do not want to create an exposed service with SOAP bindings that is automatically connected to your Mediator component.
5. In the WSDL File field, enter name of the WSDL file.

You can either use an existing WSDL file or create a new WSDL file. Click **Find Existing WSDLs** to use an existing WSDL file or **Generate WSDL From Schema(s)** to create a new WSDL file.

For more information on these options, refer to [Section 5.5, "Generating a WSDL File"](#).

6. In the Port Type list, select a port. Oracle JDeveloper 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 response is sent in asynchronous communication.
8. Click **OK**.

5.3.3 Creating a Mediator with One-Way Interface Definition

A Mediator component 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.

5.3.3.1 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.

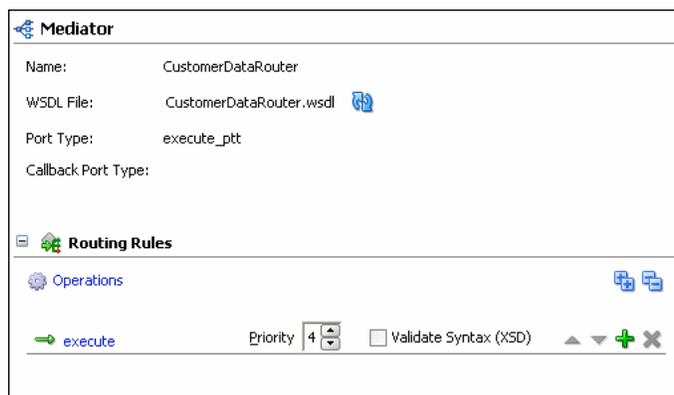
The Create Mediator dialog is displayed.

2. In the Name field, enter a name for the Mediator component.
3. In the Template list, select **One-Way Interface**.
4. Deselect the Create Composite Service with SOAP Bindings option if you do not want to create an exposed service with SOAP bindings that is automatically connected to your Mediator component 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 **OK**.

5.3.3.2 What Happens When You Create a Mediator Component with One-Way Interface Definition

A Mediator component for one-way interaction with port types defined for input message is created. [Figure 5-5](#) 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 5-5](#), represent a one-way operation.

Figure 5-4 One-Way Interface Mediator component in Mediator Editor



5.3.4 Creating a Mediator with Synchronous Interface Definition

A Mediator component 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.

5.3.4.1 How to Create a Mediator with Synchronous Interface Definition

You can create a Mediator component 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 component.
3. In the Template list, select **Synchronous Interface**.
4. Clear the Create Composite Service with SOAP Bindings option if you do not want to create an exposed service with SOAP bindings that is 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**.

5.3.4.2 What Happens When You Create a Mediator Component with Synchronous Interface Definition

A Mediator component with port types defined for 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 5-5](#) shows how a Mediator created with synchronous interface looks like in Mediator Editor. The arrows to the left of the execute operation in [Figure 5-5](#), represent a synchronous operation.

Figure 5-5 Synchronous Mediator component in Mediator Editor



5.3.5 Creating a Mediator with Asynchronous Interface Definition

A Mediator component 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.

5.3.5.1 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.

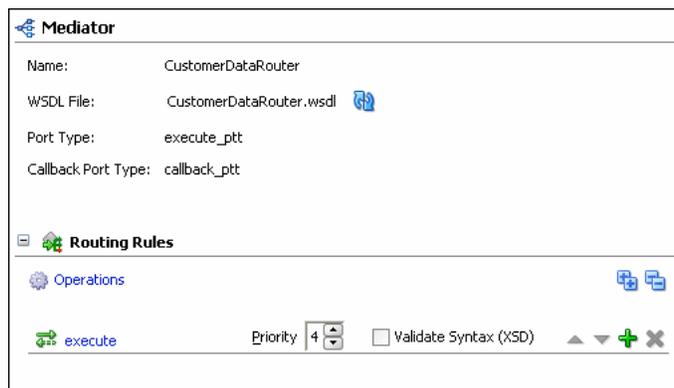
To create a Mediator with asynchronous interface definition:

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 component.
3. In the **Template** list, select **Asynchronous Interface**.
4. Deselect the Create Composite Service with SOAP Bindings option if you do not want to create an exposed service with SOAP bindings that is automatically connected to your Mediator component 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**.

5.3.5.2 What Happens When You Create a Mediator Component with Asynchronous Interface Definition

A Mediator component for asynchronous interaction with port types defined for request and response message is created. [Figure 5–6](#) 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. Callback Port Type displays the port on which the response is sent. The arrows to the left of the execute operation in [Figure 5–6](#), represent an asynchronous operation.

Figure 5–6 Asynchronous Mediator component in Mediator Editor



5.3.6 Creating a Mediator for Event Subscription

You can create a Mediator component 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.

See Also: [Chapter 8, "Business Events and the Event Delivery Network"](#) for information about business events.

5.3.6.1 How to Create a Mediator component 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 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 **Subscribe to Events**.
4. Click **Add**.

The Event Chooser dialog is displayed.

5. Click **Search** to the right of Event Definition field.

The SCA Resource Lookup dialog is displayed.

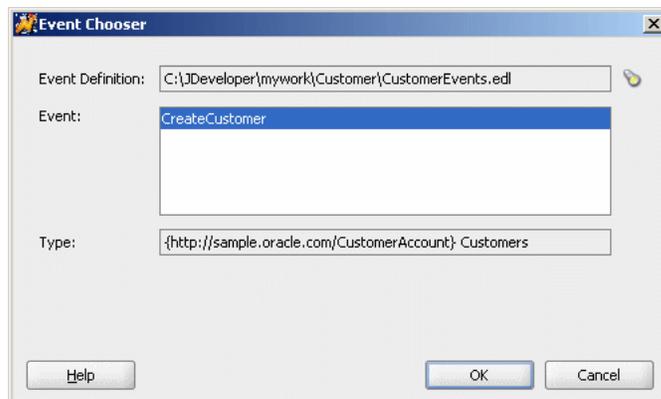
6. Select an event definition file (.edl) and click **OK**.

The Event field is populated with the events described in the .edl file that you selected.

See Also: [Chapter 8, "Business Events and the Event Delivery Network"](#) for information on how to create .edl files.

7. Select one or more events in the Event field as shown in [Figure 5–7](#), and click **OK**.

Figure 5–7 *Event Chooser Dialog*

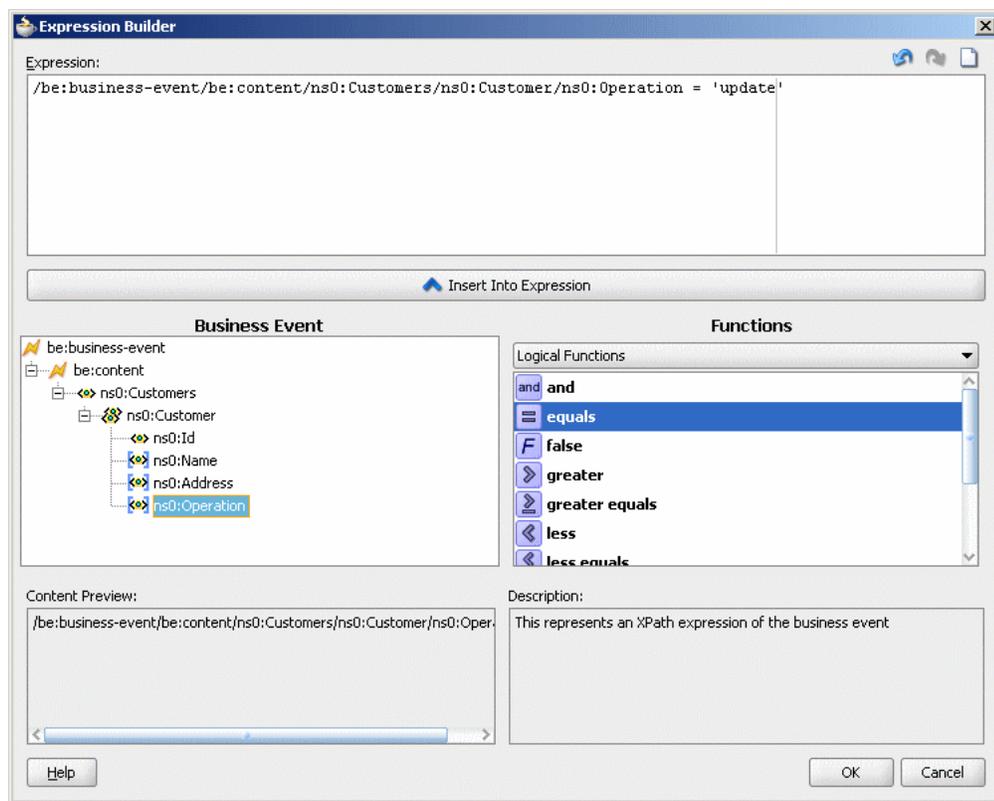


8. Select a level of delivery consistency for the event.
 - one and only one: A global (JTA) transaction is used for event delivery. If the event call fails, the transaction is rolled back and the call is retried a configurable amount of times.
 - guaranteed: A local transaction is used to guarantee delivery. There are no retries upon failure.

- immediate: Events are delivered on the same thread and on the same transaction as the caller.
- 9. Enter a security role under which an event subscription is run. By default, event subscription runs under the security of the event publisher `$publisher`.
- 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).

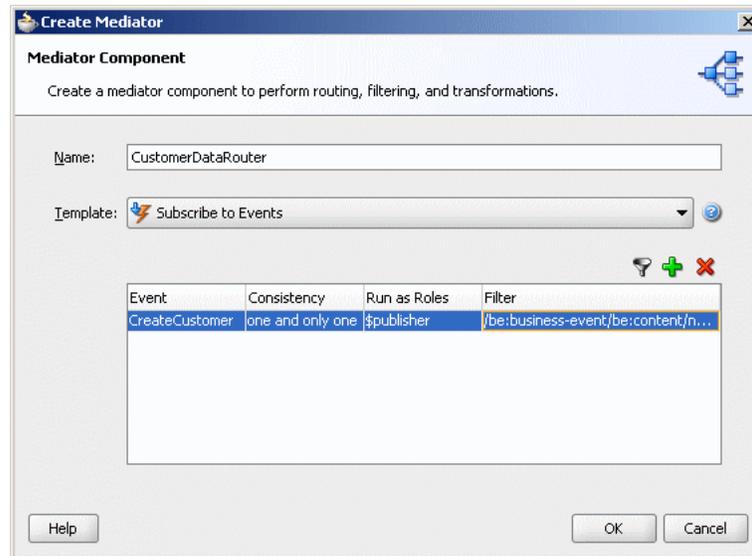
The Expression Builder dialog is displayed.
- 11. In the Expression field, enter an XPath expression and click **OK**. [Figure 5–8](#) shows a sample Expression Builder dialog box.

Figure 5–8 Business Event Filter



The Filter column of the Create Mediator dialog box is populated as shown in [Figure 5–9](#).

Figure 5–9 Create Mediator Dialog with Filter Expression



12. Click OK.

A Mediator component is created as shown in Figure 5–10. An icon on the left side indicates that Mediator component is configured for an event subscription.

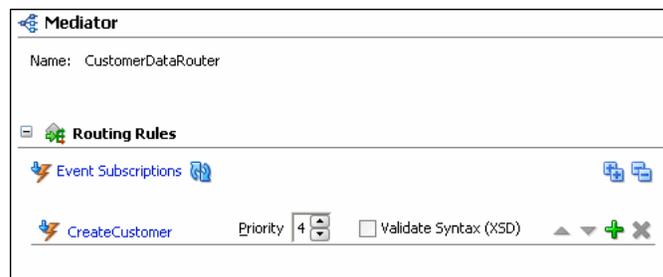
Figure 5–10 Mediator component Created with Subscribe to Events Template



13. Double-click the Mediator component.

The Mediator Editor shown in Figure 5–11 is displayed.

Figure 5–11 Subscribe to Event Mediator component in Mediator Editor



5.4 Defining Interface for an Empty Mediator Component

You can define the interface of an empty mediator by subscribing to events or by defining services.

5.4.1 Subscribing to Events

To subscribe to events:

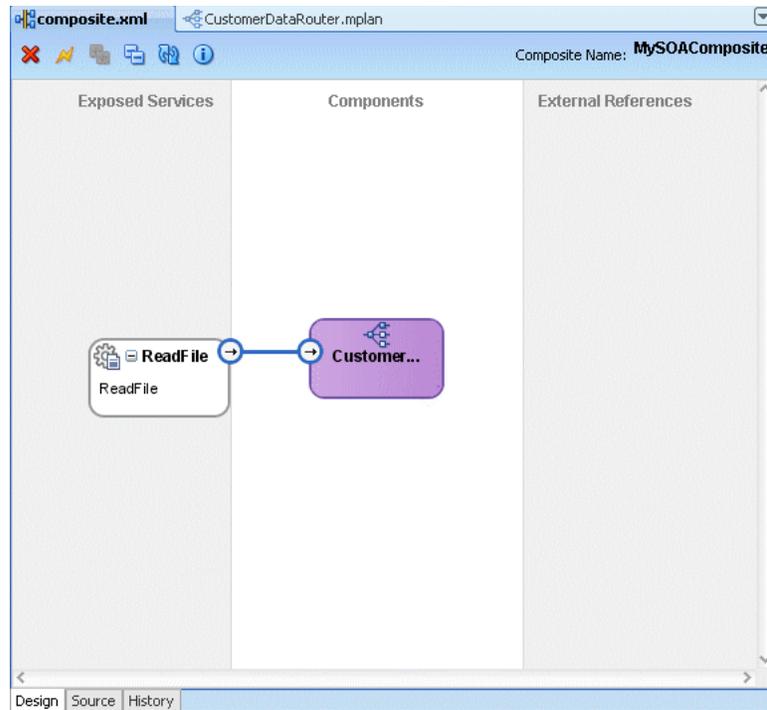
1. Double-click the Mediator component in SOA Composite Editor.
The Mediator editor is displayed.
2. Click **Add** to the right of Event Subscriptions.
The Subscribed Events dialog box is displayed.
3. Click **Add**.
The Event Chooser dialog box is displayed.
4. Click **Search** to the right of the Event 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, enter a security role under which an event subscription is run.
8. Double-click the Filter field to specify an expression for filtering the event.
9. Click **OK**.

See Also: [Section 5.3.6, "Creating a Mediator for Event Subscription"](#) for detailed information about Consistency, Run as Roles, and Filter fields of an event.

5.4.2 Defining Services for a Mediator component

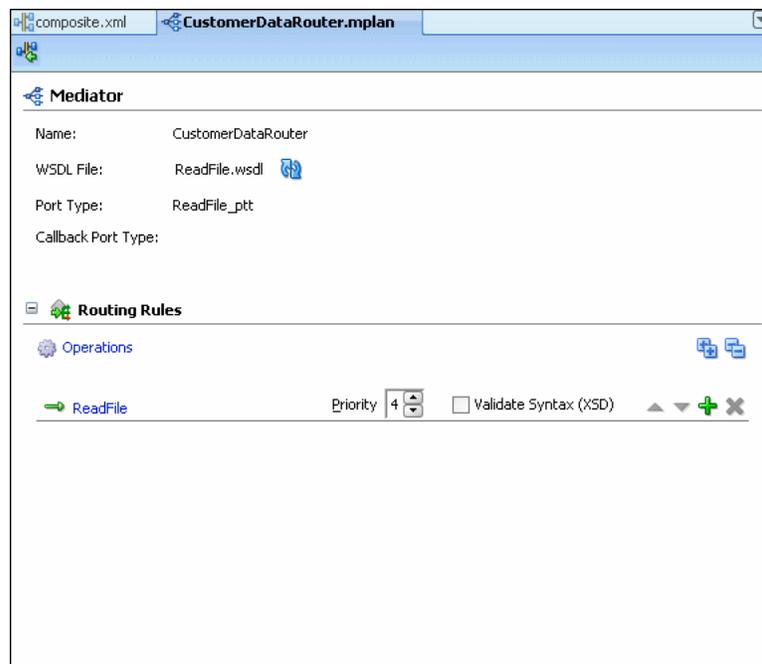
You can define service for a Mediator component by connecting the Mediator component to a service through a wire in SOA Composite Editor. The service for a Mediator component is automatically defined by using the WSDL file from the wire source. For example, if you connect the `ReadFile` service shown in [Figure 5-12](#) to the `CustomerDataRouter` mediator, then the `CustomerDataRouter` mediator automatically inherits the service definition of the `ReadFile` service.

Figure 5–12 Connecting Mediator to a Service



On double-clicking the Mediator Editor would appear as shown in [Figure 5–13](#).

Figure 5–13 Mediator Editor



You can also use the Define Service option in the Mediator Editor to define services for a mediator component.

To define service for a Mediator component in Mediator Editor:

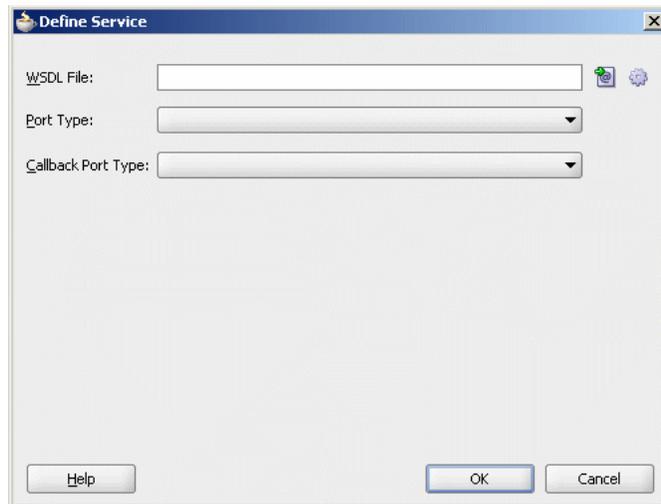
1. Double-click the Mediator component in SOA Composite Editor.

The Mediator editor is displayed.

2. Click **Add** to the right of WSDL File.

The Define Service dialog box is displayed, as shown in [Figure 5–14](#).

Figure 5–14 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.

See Also: [Section 5.5, "Generating a WSDL File"](#) for information about how to generate 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**.

5.5 Generating a WSDL File

You can generate a WSDL file from an existing XSD file or a file in a native file format such as a comma-separated value (CSV) file, a fixed-length file, a document type definition (DTD) file, or a COBOL copybook file.

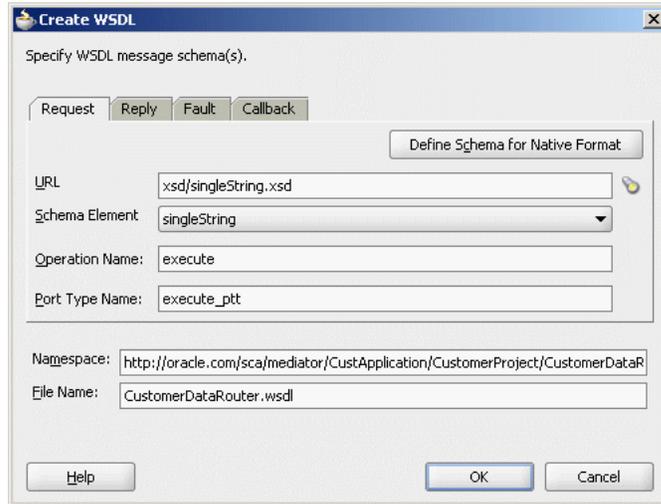
Oracle JDeveloper uses the schema files to define the request, reply, fault, and callback messages. You can specify the same or different schema files for the request, response, fault, and callback messages. Minimally, you must specify the schema for the request message.

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 an empty mediator.

Each of this method opens the Create WSDL dialog shown in [Figure 5–15](#).

Figure 5–15 Create WSDL Dialog

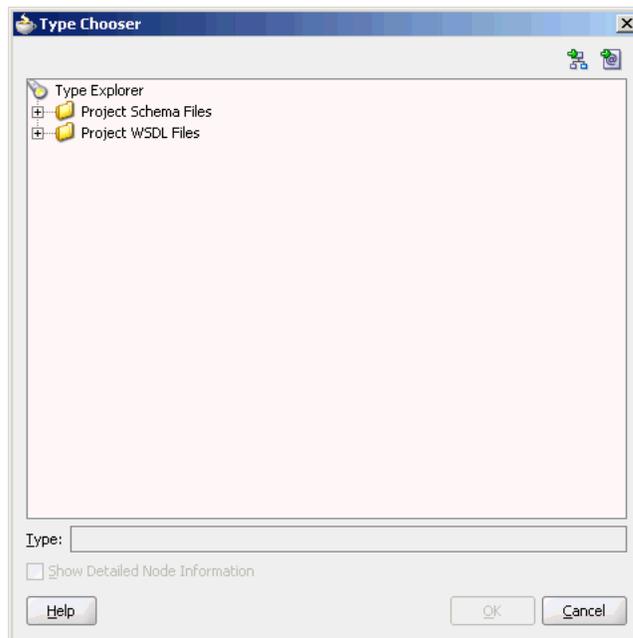


To generate a WSDL file from an existing XSD file:

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 5–16 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

The steps for generating a WSDL file from a sample file are similar to steps mentioned in "[To generate a WSDL file from an existing XSD file:](#)". The only difference is that instead of clicking the Search button in the request, response, fault, and callback tabs of Create WSDL dialog, you need to click Define Schema for Native Format button. This opens the Native Format Builder wizard. A WSDL file is generated after you complete the wizard. If you need assistance on a wizard page, then click Help. You can also refer to *Oracle Fusion Middleware User's Guide for Technology Adapters* for information about Native Format Builder wizard.

5.6 Specifying Operation or Event Subscription Properties

After creating a Mediator component, you can use the Mediator Editor to specify the following properties of an operation or event subscription.

- Priority

You can specify the priority of an event or an operation. The priority determines the order in which the services or events are processed. For example, if two events, to which a Mediator component is subscribing, are raised at the same time, then the event with higher priority is processed first.

The priority value can range from 0 to 9. The message with priority 9 has a highest priority during dequeue. The default priority is set to 4.

- Validate Syntax (XSD)

You can select this option to validate the schemas of the inbound messages. By default, validate schema is set to `false`.

5.7 Modifying a Mediator component

You can modify the operations or event subscriptions of a Mediator component by using the Mediator Editor.

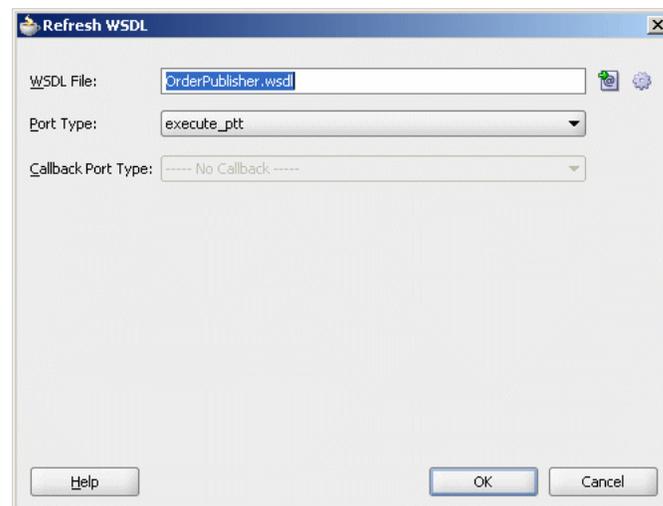
5.7.1 Modifying Mediator component Operations

Perform the following steps to modify operations of a Mediator component:

1. In Mediator Editor, click the **Refresh Operations From WSDL** icon to the right of the WSDL File field.

The Refresh WSDL dialog is displayed as shown in [Figure 5–17](#).

Figure 5–17 Refresh WSDL Dialog

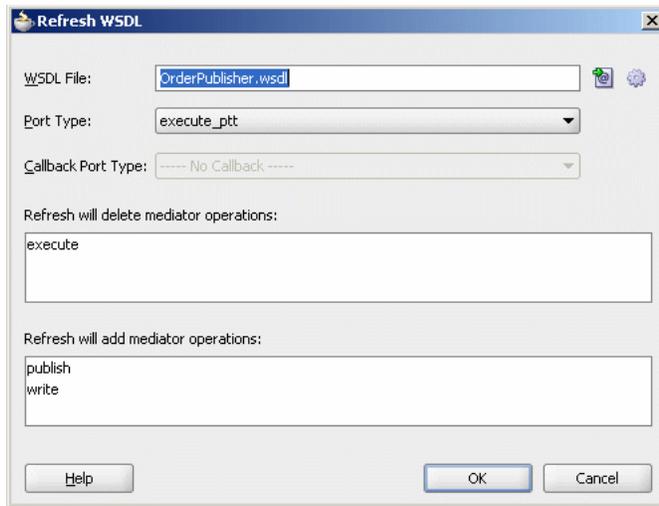


2. Specify a WSDL file in the WSDL File field.

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 5–18](#).

Figure 5–18 Refresh WSDL Dialog with Updated Operations



3. Click **OK**.
4. From the **File** menu, select **Save All**.

5.7.2 Modifying Mediator component Event Subscriptions

Perform the following steps to modify events subscription of a Mediator component:

1. In Mediator Editor, click the **Manage Event Subscriptions** icon to the right of the Event Subscriptions.

The Manage Event Subscriptions dialog is displayed as shown in [Figure 5–19](#).

Figure 5–19 Manage Event Subscriptions Dialog



2. You can perform any one 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.

See Also: [Section 5.3.6, "Creating a Mediator for Event Subscription"](#) for detailed information about Consistency, Run as Roles, and Filter fields of an event.

3. Click **OK**.
4. From the **File** menu, select **Save All**.

5.8 Deleting a Mediator component

To delete a Mediator component, perform the following steps:

1. Select the Mediator component in the SOA Composite Editor.
2. Click **Delete** at the top of SOA Composite Editor tab or right-click the Mediator component and select **Delete**.
3. Confirm that you want to delete the selected component.
4. Save your changes.

You can also delete multiple Mediator components in SOA Composite Editor. To do this, press Ctrl and select the Mediator components that you want to delete and then click **Delete**.

Note: Do not delete a Mediator component in the Application Navigator.

Creating Routing Rules

This chapter provides an overview of mediator routing rules and describes how to specify routing rules for an Oracle Mediator service component (Mediator component).

This chapter includes the following topics:

- [Section 6.1, "Introduction to Routing Rules"](#)
- [Section 6.2, "Defining Routing Rules"](#)
- [Section 6.3, "Creating CustomerRouter Mediator for Routing Messages"](#)

6.1 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 might need to be transformed. These two tasks, routing and transformations, are the core responsibilities of the Oracle Mediator. You can use the routing rules to specify how a message processed by a Mediator component reaches its next destination. Routing rules specify where a Mediator component sends the message, how it sends it, and what changes should be made to the message structure before sending it to the target service.

You can specify routing rules only if a service or an event has been defined for a Mediator component.

See Also: [Chapter 5, "Getting Started with Oracle Mediator"](#) for information about how to define services or events for a Mediator component.

When you configure routing rules, you can specify the following details:

- Target service
The service to which the message should be sent. See [Section 6.2.1.1, "Specifying Target Service"](#) for more information about how to invoke a target service.
- Filter expression
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. See [Section 6.2.1.2, "Specifying Expression for Filtering Messages"](#) for information about how to specify filter expressions.
- Execution type

Specify the way in which routing rules are executed. You can specify either of the following execution types: sequential or parallel.

See [Section 6.2.1.3, "Specifying Sequential or Parallel Execution"](#) for information about how to specify an execution type.

- Schematron based validations

Specify the schematron files for validating different parts of an inbound message.

See [Section 6.2.1.4, "Using Semantics Validation"](#) for information about how to perform schematron based validations.

- Transformations

Document 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 transform data from one XML schema to another, thus enabling data interchange among applications using different schemas. However, to set the target message properties irrespective of the source properties, payload part or constants, you can use the assign value feature. See [Section 6.2.1.5, "Creating Transformations"](#) and [Section 6.2.1.6, "Assigning Values"](#) for information about how to create transformations.

- Reply, callback, and fault handlers

You can specify how to handle synchronous reply, callback, and fault messages. See [Section 6.2.1.7, "Handling Response Messages"](#) for information about synchronous reply, callback, and fault messages handling.

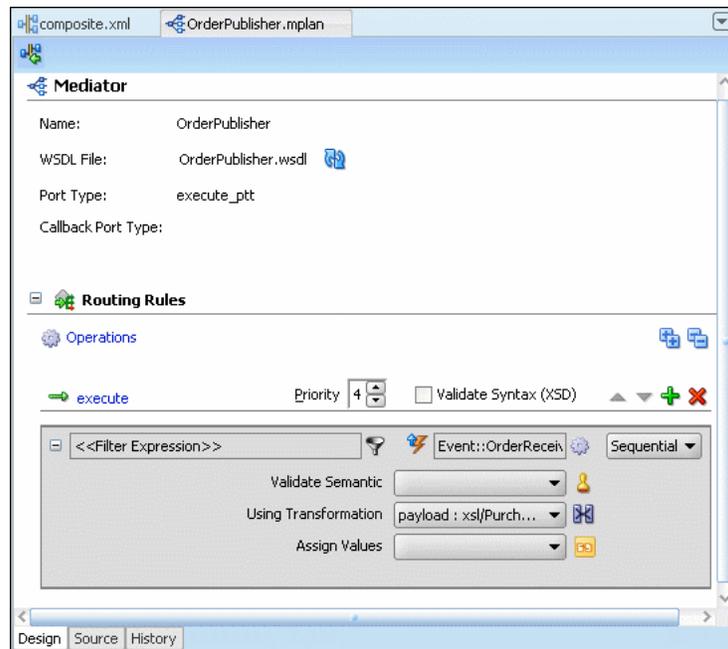
6.2 Defining Routing Rules

Routing rules can be defined only for a Mediator component which has a service or an event defined. Routing rules can be defined using Oracle JDeveloper. To access the Mediator Editor, use either of the following two methods.

- From the SOA Composite Editor:
 - a. Double-click the icon that represents the Mediator component 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 component for which you want to specify the routing rules. The Mediator component file has `mplan` extension.
 - c. Click the Plus (+) icon next to the Routing Rules panel.

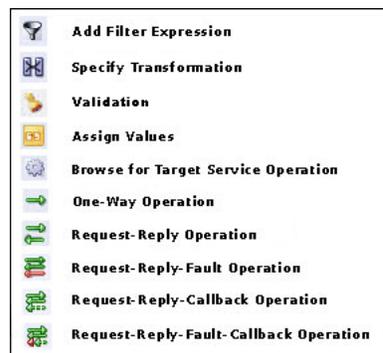
[Figure 6–1](#) shows the Routing Rules panel in Mediator Editor.

Figure 6–1 Mediator Editor- Routing Rules Panel



The icons in the Routing Rules panel are summarized in [Figure 6–2](#).

Figure 6–2 Routing Rule Panel Icons



6.2.1 Creating Routing Rules

You can create the following routing rules for a service or event subscription:

- [Specifying Target Service](#)
- [Specifying Expression for Filtering Messages](#)
- [Specifying Sequential or Parallel Execution](#)
- [Using Semantics Validation](#)
- [Creating Transformations](#)
- [Assigning Values](#)
- [Handling Response Messages](#)

6.2.1.1 Specifying Target Service

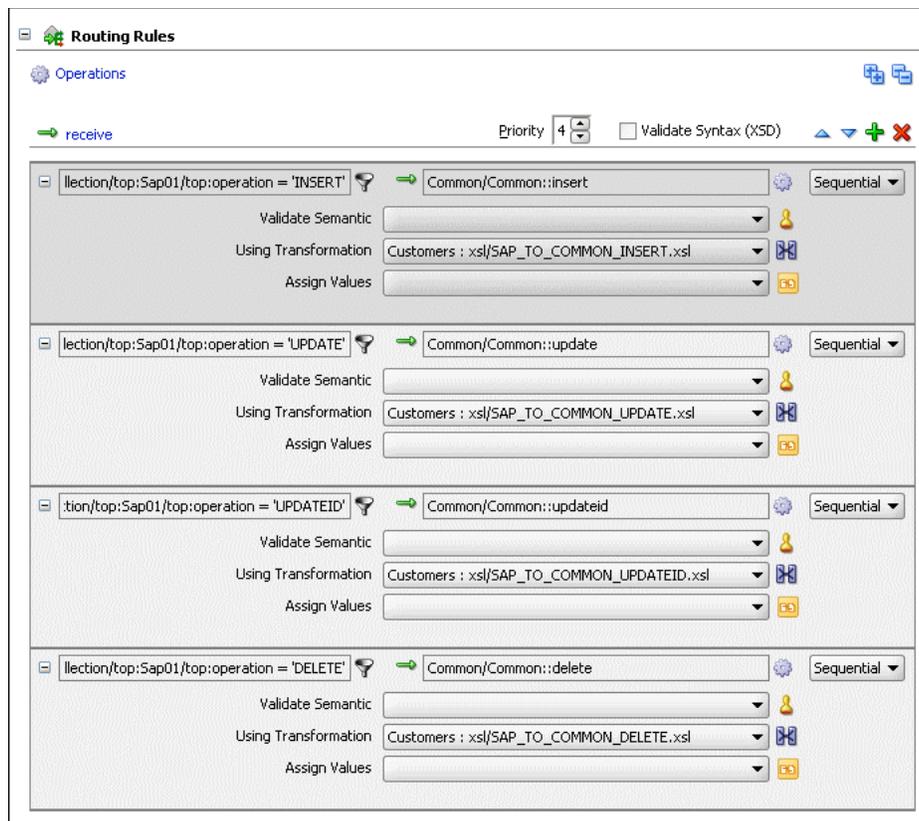
A target service specifies the next service to which a mediator should send the message and the operation to perform on that message when it reaches the target service.

You can specify multiple routings to one inbound operation or event. Each routing is mapped to one target service invocation or event. Therefore, if you want to specify multiple service invocations or raise multiple events, you must specify one routing rule for each target service operation. For example, based on a message payload, you want to invoke an operation from the following operations defined in a service:

- insert
- update
- updateid
- delete

You need to create four routings, one for each operation. Later, when you specify a filter expression, you can specify which target service and operation is applied to each message instance on the basis of the message payload as shown in [Figure 6-3](#).

Figure 6-3 Multiple Routings for an Inbound Operation



How to Invoke a Service

Perform the following steps in Mediator Editor to specify the service to be invoked by the routing rule:

1. In the Routing Rules panel, click **Add**.

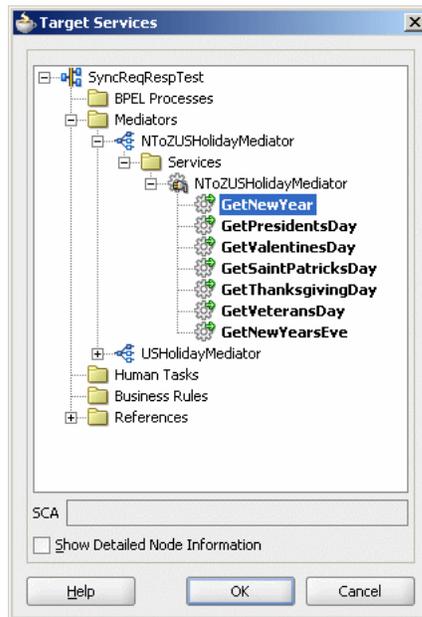
The Target Type dialog is displayed as shown in [Figure 6-4](#).

Figure 6–4 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 6–5](#).

Figure 6–5 Target Services Dialog



Note: A service can consist of multiple operations as shown in [Figure 6–5](#).

4. Click **OK**.

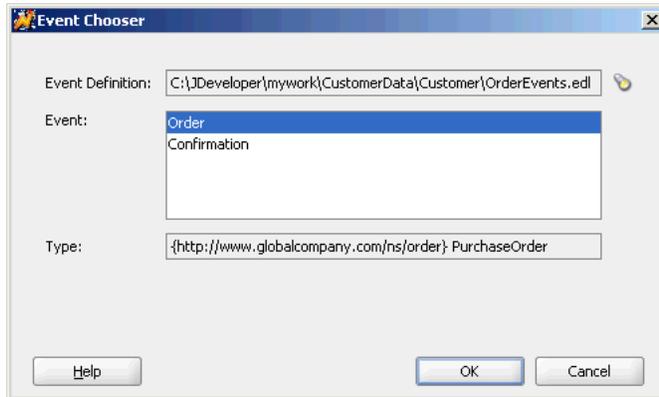
How to Raise an Event

Perform the following steps to specify the event to be raised by the routing rule:

1. In the Routing Rules panel, click **Add**.
The Target Type dialog is displayed as shown in [Figure 6–6](#).
2. Click **Event**.
The Event Chooser dialog is displayed.
3. Click **Search** to the right of Event Definition field.
The SCA Resource Lookup 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 6-6](#).

Figure 6-6 Event Chooser Dialog



5. Select an event.
6. Click **OK**.

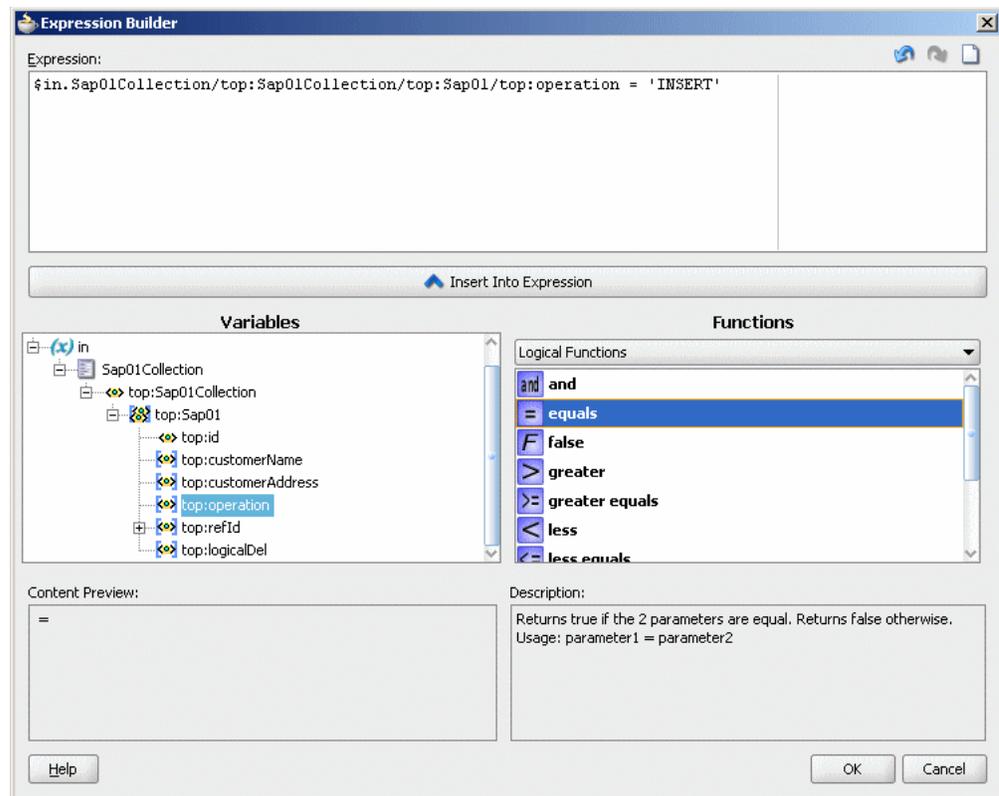
6.2.1.2 Specifying Expression for Filtering Messages

The filter expression routing rule enables you to filter messages based on their payload. If the expression filter for a given message instance evaluates to true, then the message is not delivered to the target service/operation pair specified within the routing rule.

For example, you want notices of new product launches from headquarters to be routed to three different stores: one in New York, one in Houston, and one in San Francisco. However, you only want notices regarding the product line of type **MOBILE** to be sent to the New York store. To implement this, you need to define a routing rule for each component/operation pair that sends messages to the target stores. In addition, for the routing rule that send messages to the New York store, you specify a filter expression.

You can specify a filter expression by using the Expression Builder dialog as shown in [Figure 6-7](#). This dialog is displayed when you click the icon to the right of the filter expression field in the routing rules panel.

Figure 6–7 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 component. Oracle JDeveloper parses the Mediator component 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.

An input message can consists of multiple parts. You can use `$in.<partname>` to access a part of an input message.
- **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 provides a description of a value selected from the Variables field or Functions palette.

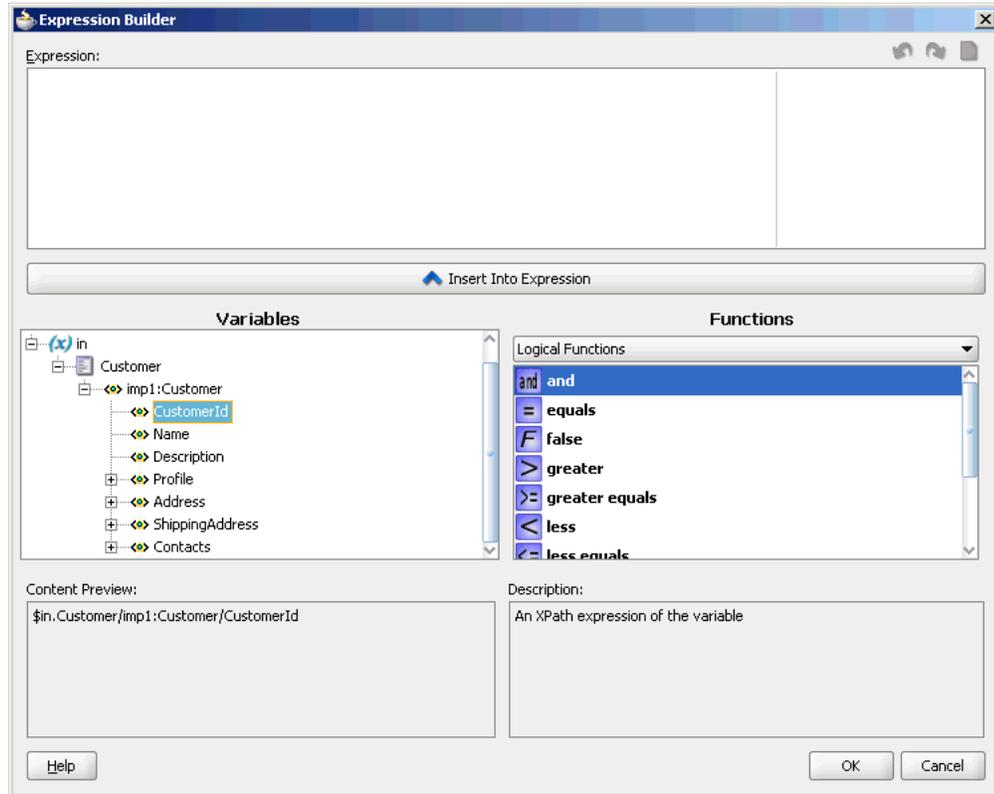
To specify a filter expression on a message payload, follow these steps:

1. In the Routing Rules panel, click the **Add Filter Expression** icon, shown in [Figure 6-2](#).

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 6-8](#).

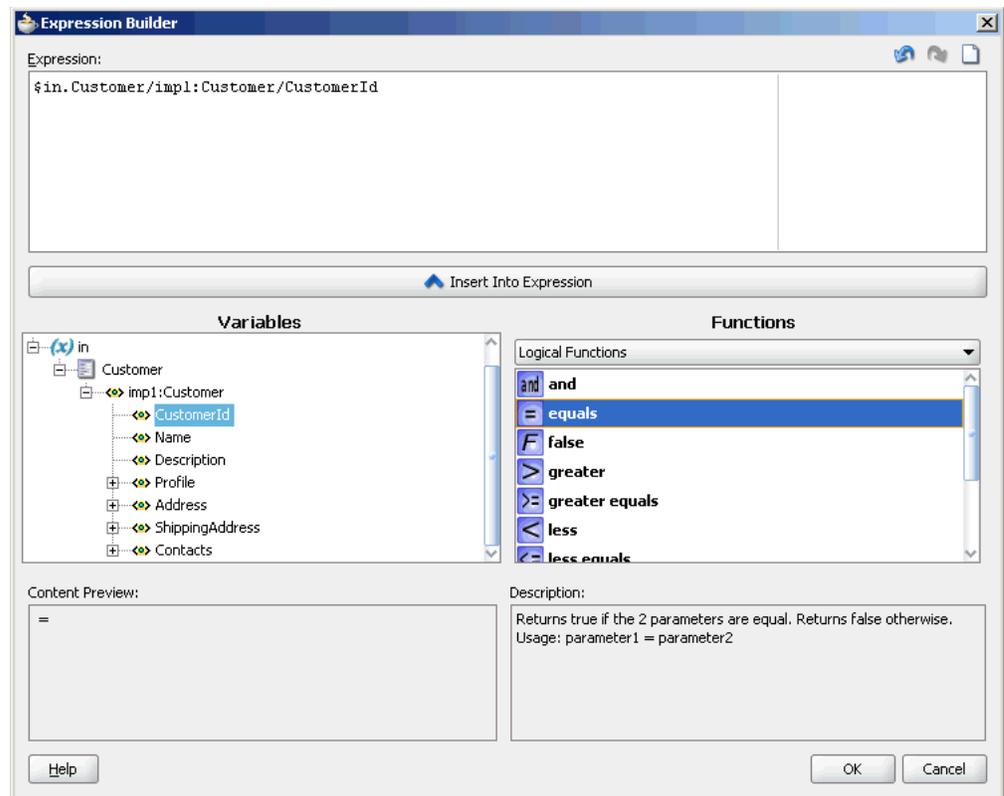
Figure 6-8 Expression Builder Dialog – Variables Element Selected



3. Click **Insert Into Expression**.

The expression is added in the Expression field, as shown in [Figure 6-9](#).

Figure 6–9 Expression Builder Dialog – Variables Element Inserted



4. From the **Function** list, select the function that you want to apply to the message payload. For example, `equals`.

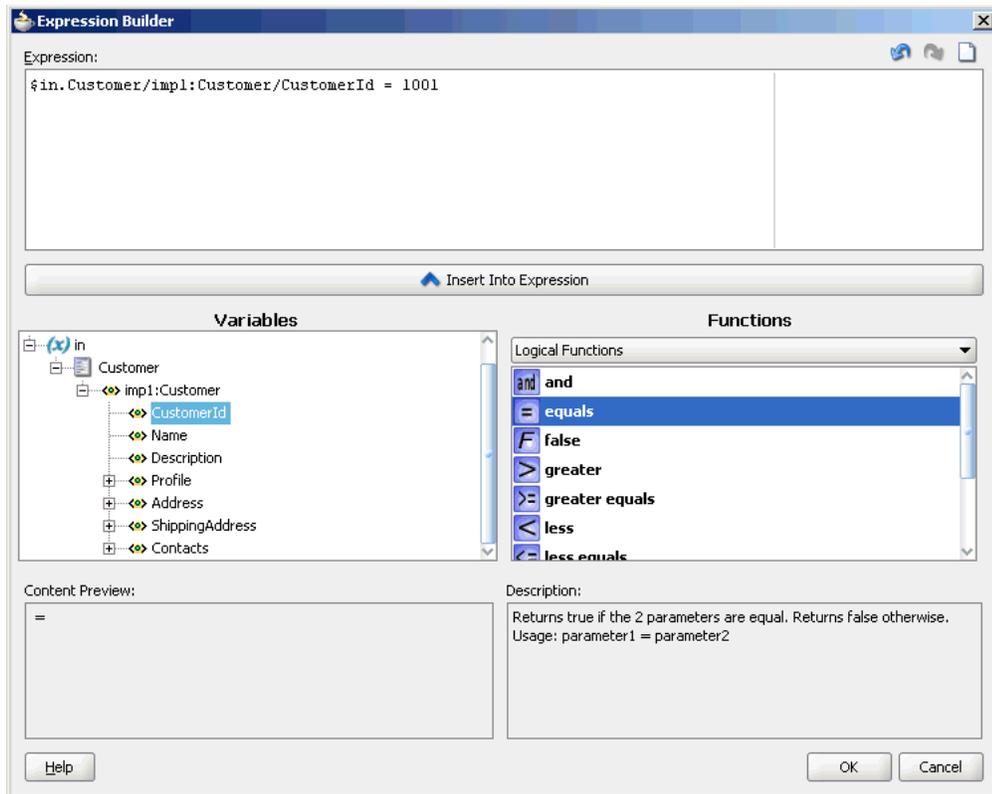
Functions are grouped in categories that are listed when you click the down arrow in the Functions list. For example, if you click the down arrow and select Logical Functions, the list appears as shown in Figure 6–9. When you select a function within the Logical Functions list, a description of that 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.

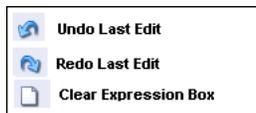
6. Complete the expression. In this example, a value of 1001 is entered, as shown in Figure 6–10.

Figure 6–10 Sample Expression Builder Dialog – Value Entered



7. You can edit the expression manually, or use the expression editing icons, which are summarized in [Figure 6–11](#).

Figure 6–11 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.

6.2.1.3 Specifying Sequential or Parallel Execution

You can specify execution type for a routing rule. A routing rule execution type can be parallel or sequential.

In sequential execution, routings are evaluated and actions are performed sequentially. The caller is blocked during the sequential execution. Sequential routings are evaluated in the same thread and transaction of the caller.

In parallel execution, routings are queued and evaluated in parallel in different threads. The caller is not blocked during the parallel execution.

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.

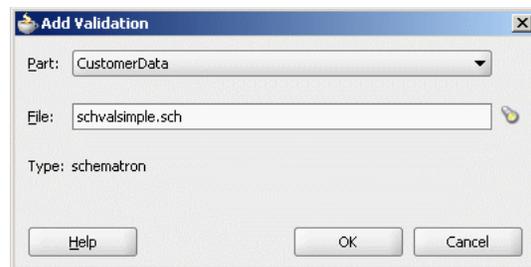
To specify an execution type for a routing rule, select Sequential or Parallel execution type from the Routing Rules panel.

6.2.1.4 Using Semantics Validation

An inbound message can consist of multiple parts. You can specify schematron files for validating an inbound message and its various parts. A schematron file has .sch extension and can be used in the following way:

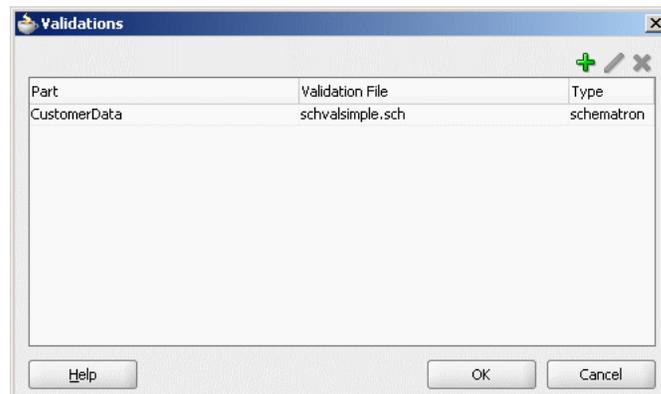
1. Click the **Select Validation File** icon to the right of Validate Semantics field.
The Validations dialog is displayed.
2. Click **Add**.
The Add Validation dialog is displayed.
3. From the **Part** list, select a message part.
4. Click **Search** to the right of the File field.
The SCA Resource Lookup dialog is displayed.
5. Select a schematron file and click **OK**.
The Add Validation dialog is updated, as shown in [Figure 6–12](#).

Figure 6–12 Add Validation Dialog



6. Click **OK**.
The Validation dialog is updated, as shown in [Figure 6–13](#).

Figure 6–13 Validation Dialog



- Click **Add** to specify a schematron file for another message part or click **OK**.

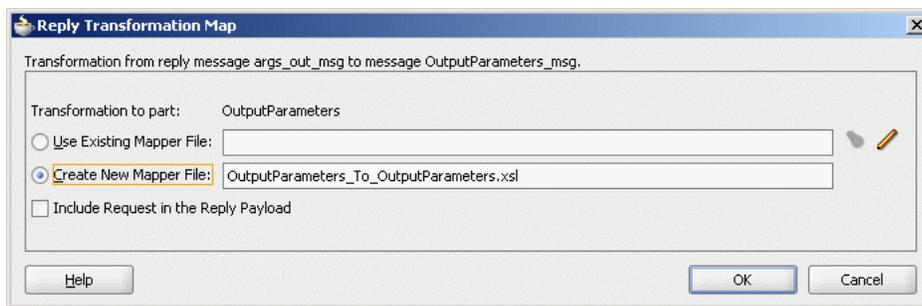
6.2.1.5 Creating Transformations

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 to another. This enables data interchange among applications using different schemas. For example, you can map incoming source purchase order schema to an outgoing invoice schema. After you define an xsl file, you can reuse it in multiple routing rule specifications.

When you click the transformation map icon to the right of the Transformation Map 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.

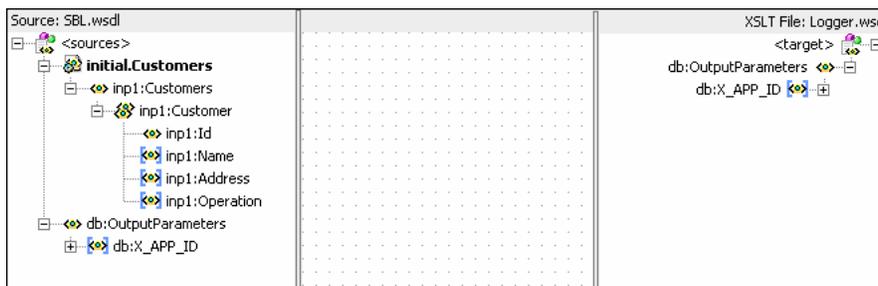
You can also specify transformations for a synchronous reply, callback response message, or a fault message. In case of synchronous reply or fault message, the Reply Transformation Map dialog or the Fault Transformation Map dialog contains the Include Request in the Reply Payload option. [Figure 6–14](#) shows a Reply Transformation Map dialog with this option.

Figure 6–14 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 6–15](#).

Figure 6–15 Initial Variable in XSL File



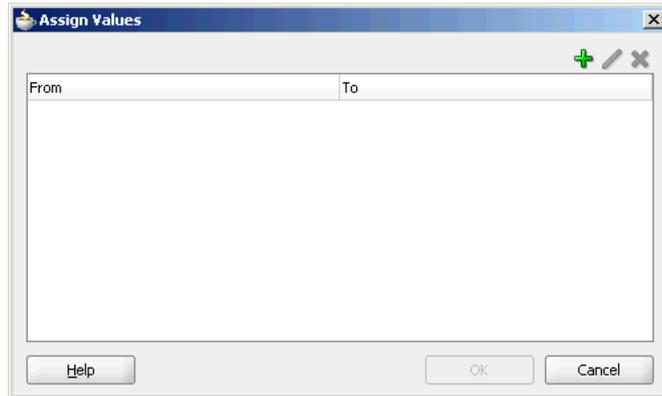
An initial message can also consists of multiple parts. You can use `$initial.<partname>` to access a part of the initial message.

For information about the Data Mapper tool, see [Chapter 4, "XSLT Mapper and Transformations"](#).

6.2.1.6 Assigning Values

You can use the assign value field to specify the properties of a target message. [Figure 6–16](#) shows the Assign Values dialog that is displayed when you click the Assign Values icon in the routing rules panel.

Figure 6–16 Assign Values Dialog

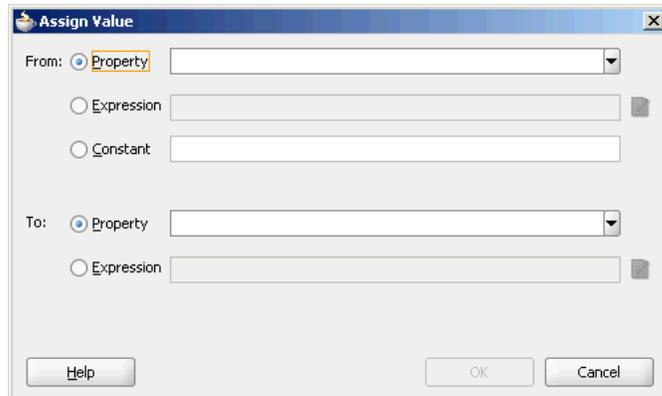


To set the properties of the target message, perform the following tasks:

1. Click **Add** in the Assign Values dialog.

The Assign Value dialog is displayed as shown in [Figure 6–17](#).

Figure 6–17 Assign Value Dialog



2. In the From section, select any of the following options:

- **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 6–7](#) is displayed.

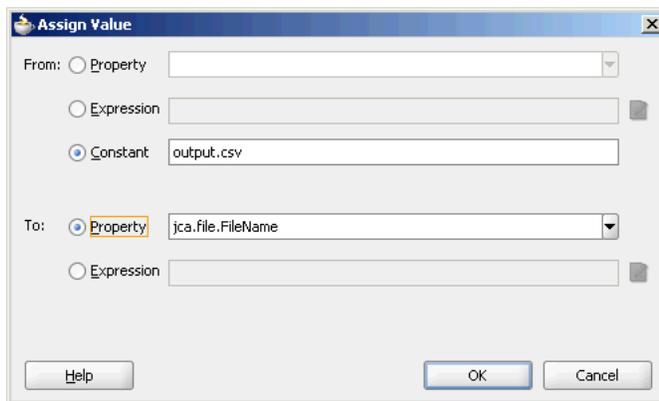
See Also: [Section 6.2.1.2, "Specifying Expression for Filtering Messages"](#) for information about Expression Builder dialog.

- **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.
 - 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 `$out` variable which contains the output message. You can use `$out.properties` to access properties of an output message and `$out.<partname>` to access a part of an output message.

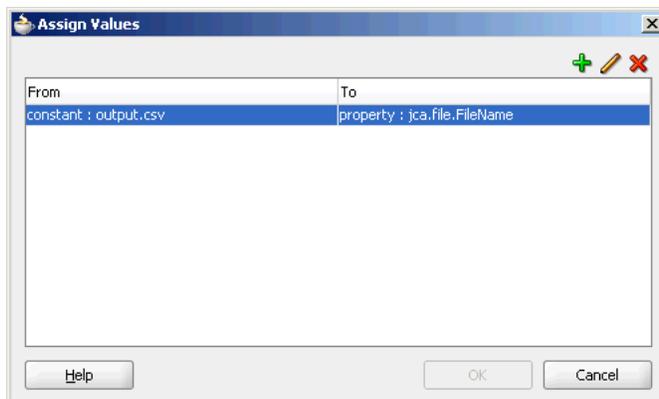
Figure 6–18 shows a sample Assign Value dialog in which a constant value `output.csv` is specified as the name for an outbound file written by a file adapter service.

Figure 6–18 Populated Assign Value Dialog



4. Click **OK** in the Assign Value dialog. The Assign Values dialog is populated as shown in Figure 6–19.

Figure 6–19 Populated Assign Values Dialog



5. Click **OK**. The expression is added to Assign Values field of the Routing Rules panel.

6.2.1.7 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.

In case of asynchronous interaction, you can specify a timeout period by when the response should come. 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 specified timeout period then a timeout response can be forwarded to another service, event, or back to the initial caller. This can be done by performing the following steps:

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.

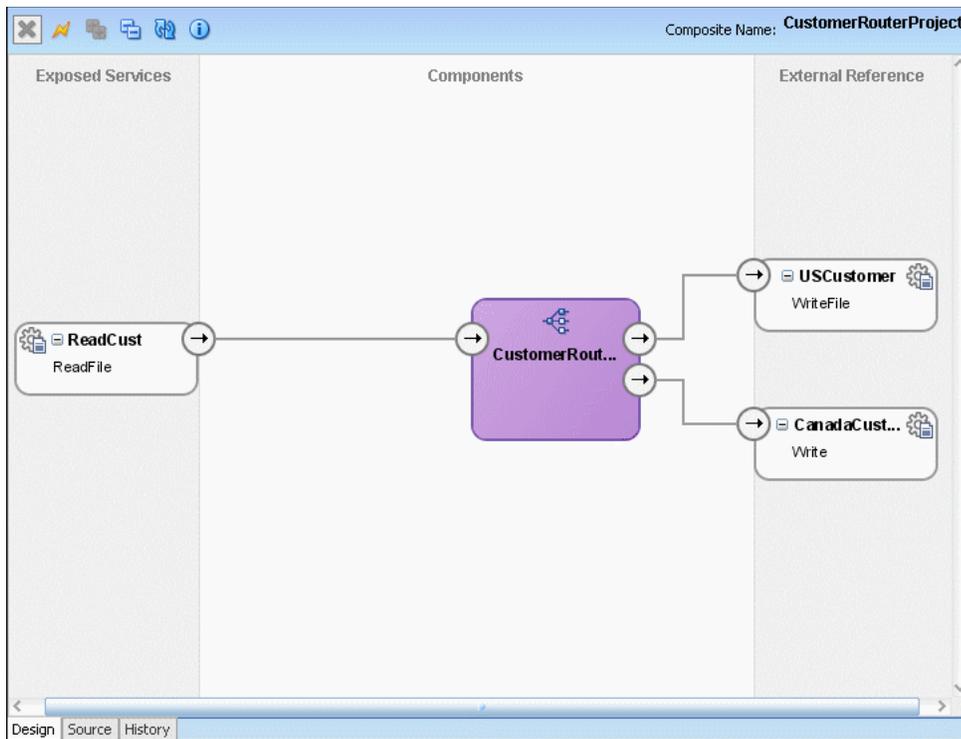
6.3 Creating CustomerRouter Mediator for Routing Messages

This section provides step-by-step instructions on creating and running the CustomerRouter use case that provides an overview of how to use a Mediator in a SOA composite application to route messages based on the payload. 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 6–20](#) provides an overview of the CustomerRouter use case.

Figure 6–20 Overview of CustomerRouter Use Case



6.3.1 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.

- [Creating an Oracle JDeveloper Application and Project](#)
- [Creating CustomerRouter Mediator](#)
- [Creating a File Adapter Service](#)
- [Creating External References](#)
- [Specifying Routing Rules](#)
- [Configuring Oracle Application Server Connection](#)
- [Deploying CustomerRouterProject](#)

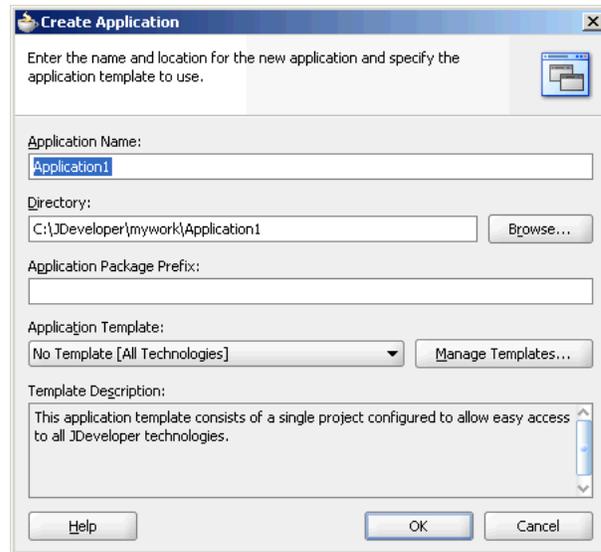
6.3.1.1 Creating an Oracle JDeveloper Application and Project

To create an application and a project for the use case, follow these steps:

1. Start Oracle JDeveloper.
2. In the upper left panel, click the **Applications Navigator** tab.
3. Right-click **Applications**, then select **New Application**.

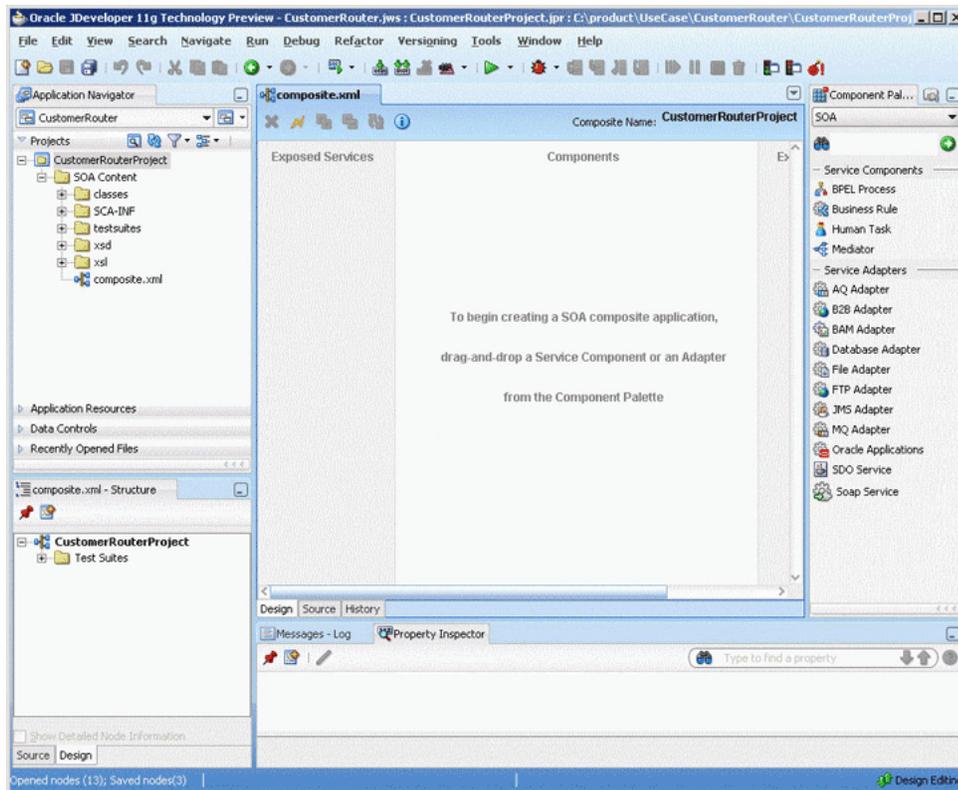
The Create Application dialog is displayed, as shown in [Figure 6–21](#).

Figure 6–21 The Create Application Dialog



4. In the **Application Name** field, enter `CustomerRouter`, and then click **OK**.
The Create Project dialog is displayed.
5. In the **Project Name** field, enter `CustomerRouterProject` and click **OK**.
6. Right-click in the Applications Navigator pane and select **New**.
The New Gallery dialog is displayed.
7. From the **Categories** navigator, select **SOA Tier**.
8. From the **Items** list, select **SOA Composite**.
9. Click **OK**.
The Create SOA Composite dialog is displayed.
10. From the Composite Template list, select **Empty Composite** and then click **OK**.
Oracle JDeveloper is displayed, as shown in [Figure 6–22](#). The Applications Navigator is updated with the new application and project and the Design Tab contains, a blank palette.

Figure 6–22 Oracle JDeveloper – Application and SOA Project Added



11. From the **File** menu, click **Save All**.

6.3.1.2 Creating CustomerRouter Mediator

Perform the following steps 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.

6.3.1.3 Creating a File Adapter Service

Perform the following steps to create a file adapter service, named `ReadCust` to read the XML files from a directory:

1. From the Components Palette, select **SOA**.
2. Select **File Adapter** and drag it to the Exposed Services design area.

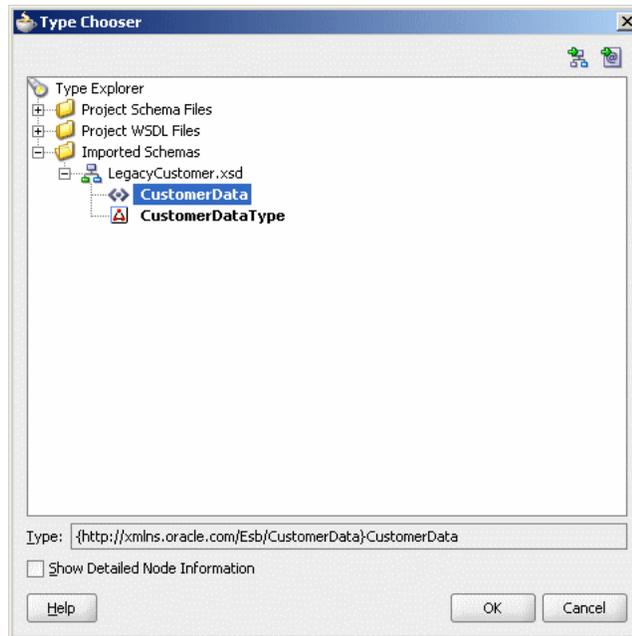
The Adapter Configuration wizard Welcome page is displayed.

3. Click **Next**.

The Service Name page is displayed.

-
4. In the Service Name field, enter `ReadCust`.
 5. Click **Next**.
The Operation page is displayed.
 6. In the **Operation Type** field, select **Read File**.
 7. In the **Operation Name** field, replace **Read** with `ReadFile`.
 8. Click **Next**.
The File Directories page is displayed.
 9. 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`.
 10. Click **Next**.
The File Filtering page is displayed.
 11. In the **Include Files with Name Pattern** field, enter `*.xml`, and then click **Next**.
The File Polling page is displayed.
 12. Change the **Polling Frequency** field value to **10 seconds**, and then click **Next**.
The Messages page is displayed.
 13. Click **Search**.
The Type Chooser dialog is displayed.
 14. Click **Import Schema File**.
The Import Schema File dialog is displayed.
 15. Click **Search** and select the `LegacyCustomer.xsd` file present in the `Samples` folder.
 16. Click **OK**.
 17. Expand the navigation tree to **Type Explorer\Imported Schemas\LegacyCustomer.xsd** and select **CustomerData**, as shown in [Figure 6-23](#).

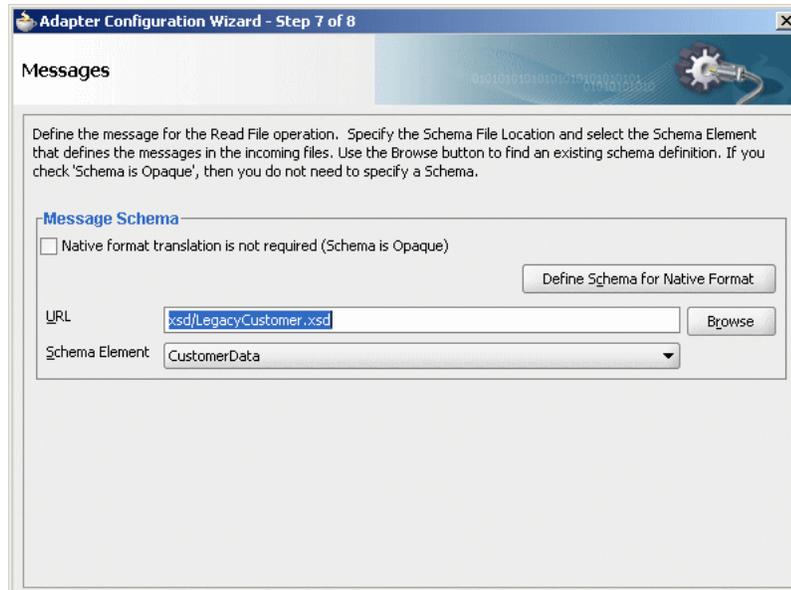
Figure 6–23 Type Chooser - CustomerData



18. Click **OK**.

The Adapter Configuration wizard appears as shown in [Figure 6–24](#).

Figure 6–24 Adapter Configuration Wizard – Messages page



19. Click **Next**.

The Finish page is displayed.

20. Click **Finish**.

21. From the **File** menu, click **Save All**.

6.3.1.4 Creating External References

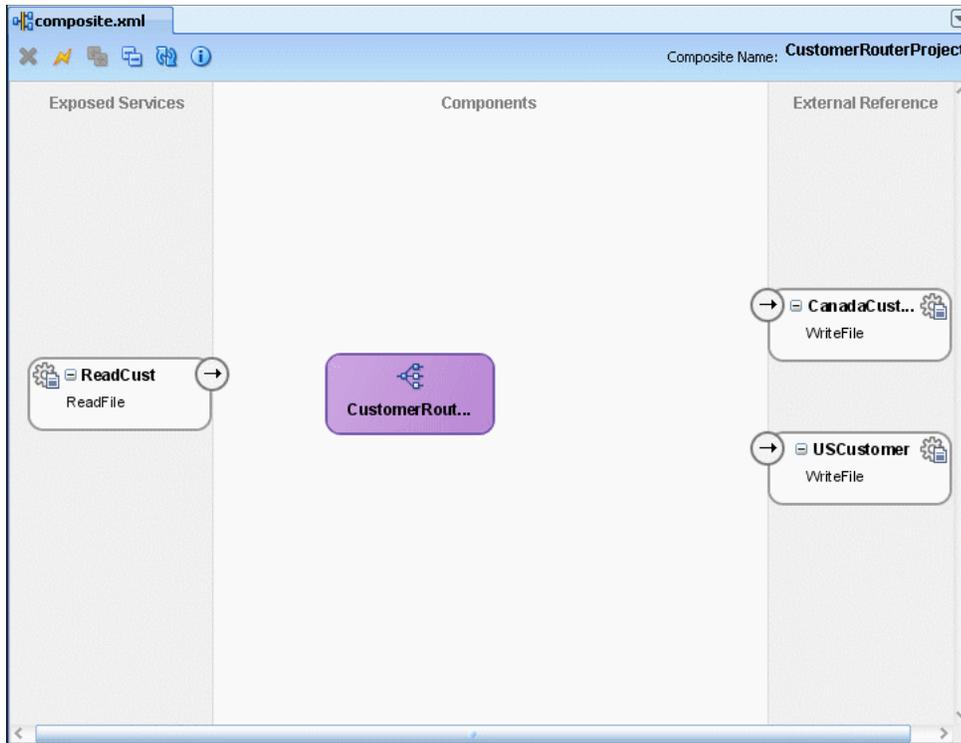
Perform the following steps to create a file adapter reference, named `USCustomer`:

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 `USCustomer`.
5. Click **Next**.
The Operation page is displayed.
6. In the Operation Type field, select **Write File**.
7. In the Operation Name field, enter `writeFile`.
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.
For example, `C:\Customer\out`.
10. In the **File Naming Convention** field, enter `customer_%SEQ%.xml` and 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 `USCustomer.xsd` file present in the `Samples` folder.
14. Click **OK**.
15. Expand the navigation tree to **Type Explorer\Imported Schemas\USCustomer.xsd** and then select **Customer**.
16. Click **OK**.
17. Click **Next**.
The Finish page is displayed.
18. Click **Finish**.
19. From the **File** menu, click **Save All**.

Create another file adapter reference `CanadaCustomer` in similar way by using the `CanCustomer.xsd` file.

[Figure 6–25](#) shows how the SOA composite editor appears after performing this task.

Figure 6–25 Mediator Component with Adapter Services and References



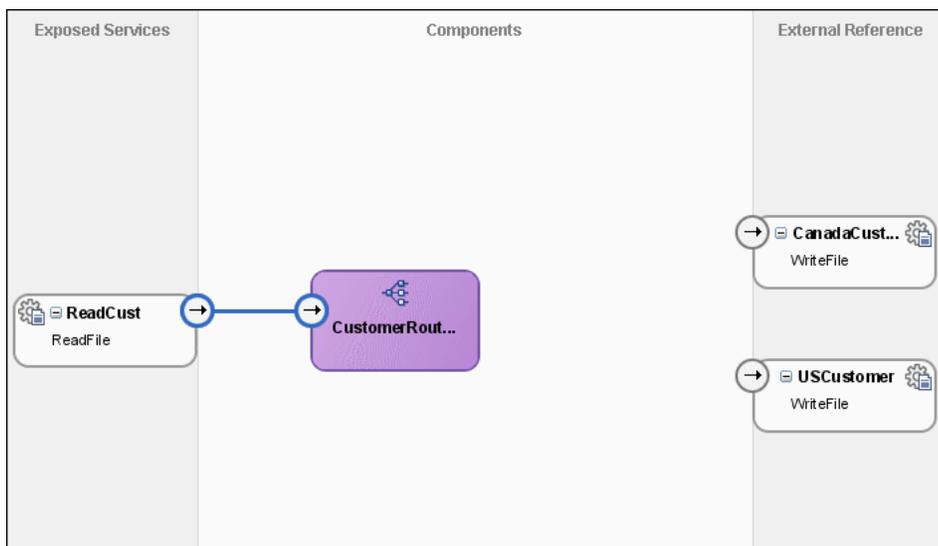
6.3.1.5 Specifying Routing Rules

Follow these steps to specify the path that messages take from the ReadCust adapter service to external references:

1. Connect the ReadCust service to the CustomerRouter mediator as shown in [Figure 6–26](#).

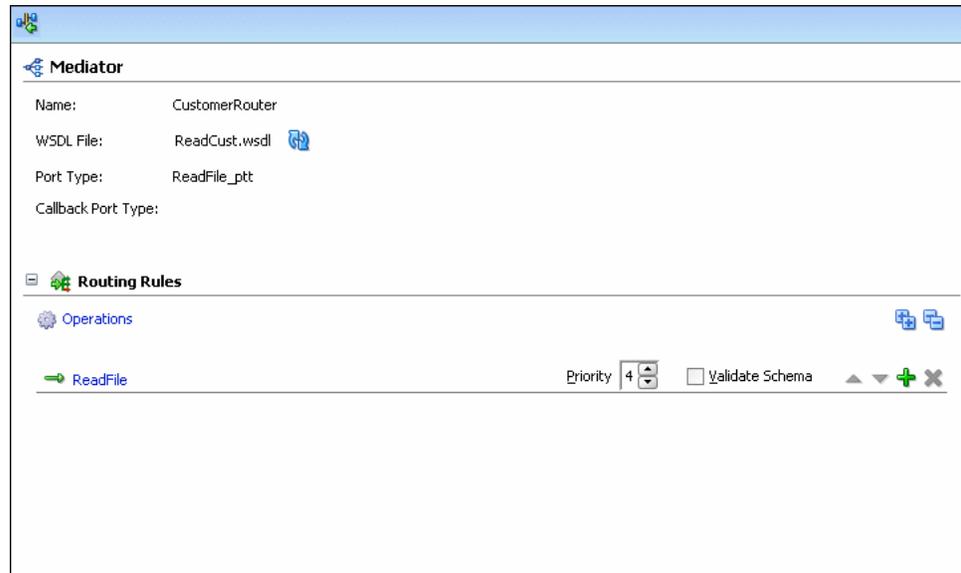
This specifies the file adapter service to invoke the CustomerRouter mediator while reading a file from the input directory.

Figure 6–26 Connecting ReadCust Service to the CustomerRouter Mediator



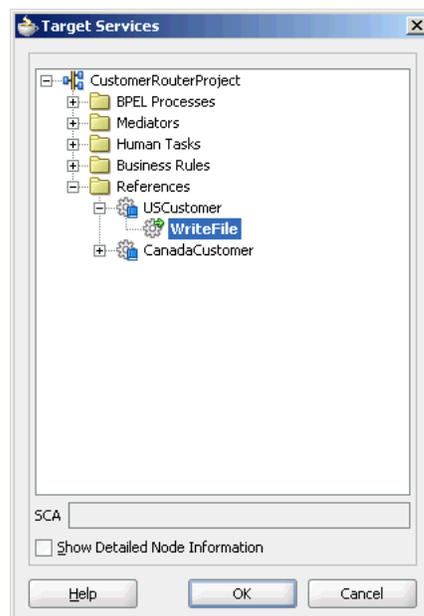
2. Double-click **CustomerRouter** mediator to open the `CustomerRouter.mplan` editor shown in [Figure 6-27](#).

Figure 6-27 *CustomerRouter Mediator in Mediator Editor*



3. In the Routing Rules section, click **Add** to the extreme right side of `ReadFile`. The Target Type dialog is displayed.
4. Select **Service**. The Target Services dialog is displayed.
5. Navigate to **CustomerRouterProject**, **References**, **USCustomer** and select **WriteFile** as shown in [Figure 6-28](#).

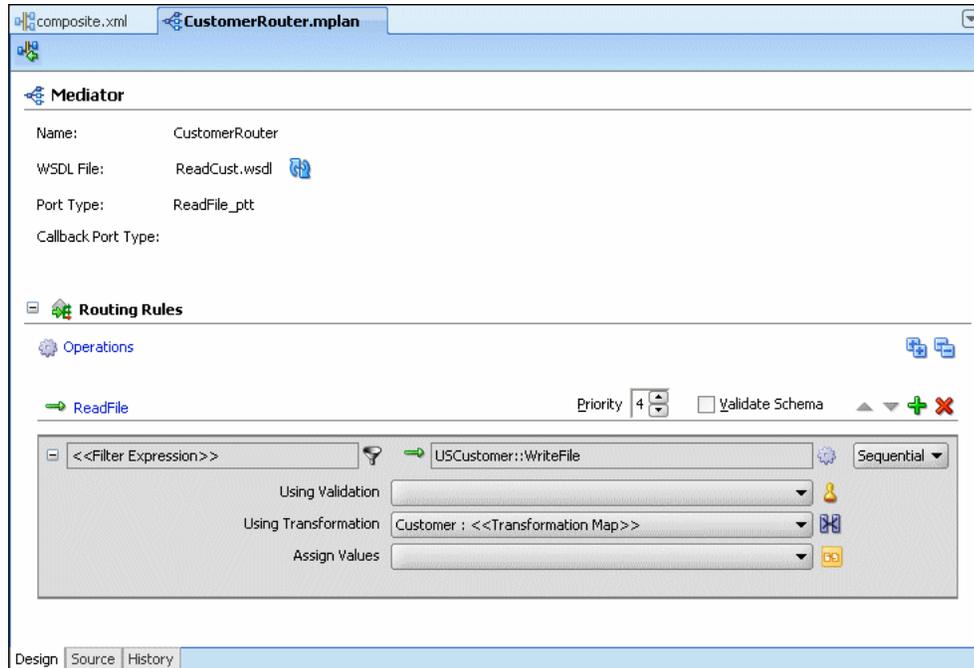
Figure 6-28 *Target Services Dialog*



6. Click **OK**.

The Routing Rules panel is displayed, as shown in [Figure 6–29](#).

Figure 6–29 The Routing Rules Panel - MapCustomerData Added



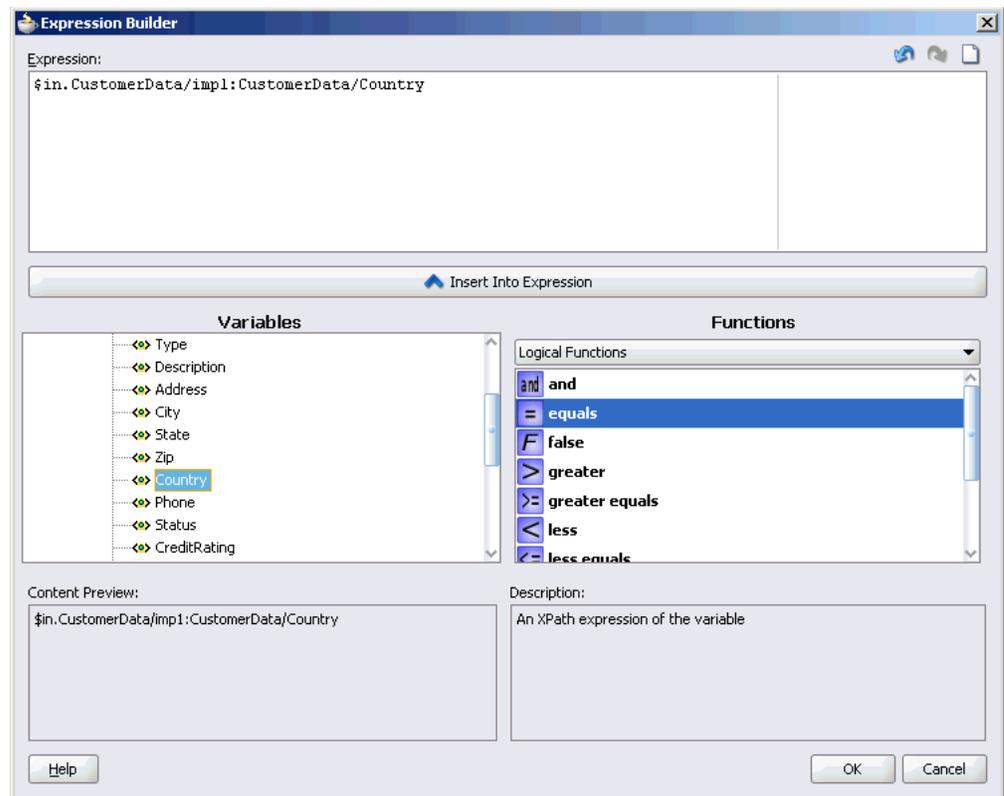
7. Click the filter icon next to the <<Filter Expression>> field to create a filter expression for this routing rule.

The Expression Builder dialog is displayed.

8. In the Variables field, navigate to **Variables, in, CustomerData** and then select **Country**.
9. Double-click **Country**.

The Country node is added in the Expression text field as shown in [Figure 6–30](#).

Figure 6–30 Expression Builder Dialog



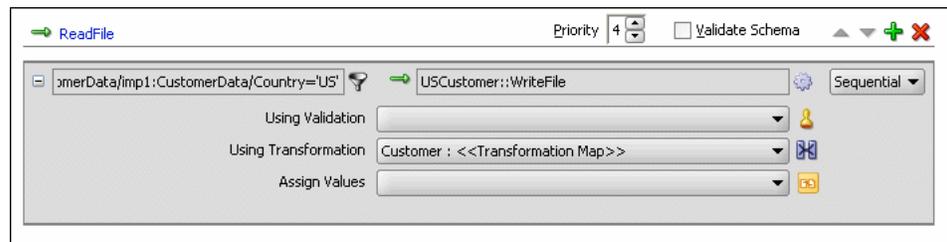
10. Modify the expression to the following:

```
$in.CustomerData/impl:CustomerData/Country='US'
```

11. Click **OK**.

The filter field of the Routing Rules panel is populated with the expression as shown in [Figure 6–31](#).

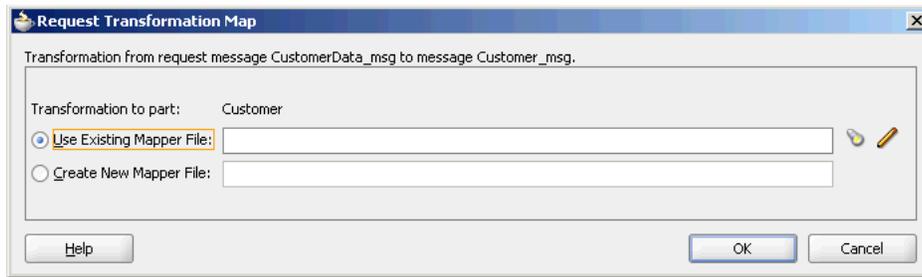
Figure 6–31 Populated Filter Field of Routing Rules Panel



12. Click the icon to the right of the Using Transformation field.

The Request Transformation Map dialog is displayed, as shown in [Figure 6–32](#).

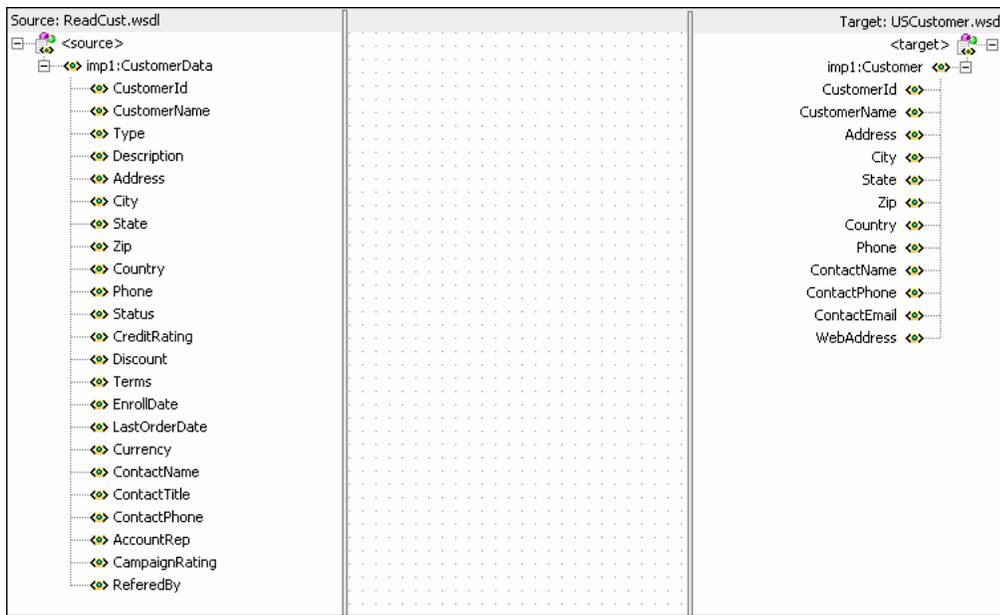
Figure 6–32 Request Transformation Map



13. Select **Create New Mapper File** and click **OK**.

A CustomerData_To_Customer.xsl tab is added, as shown in [Figure 6–33](#).

Figure 6–33 CustomerData_To_Customer.xsl Tab – Initially



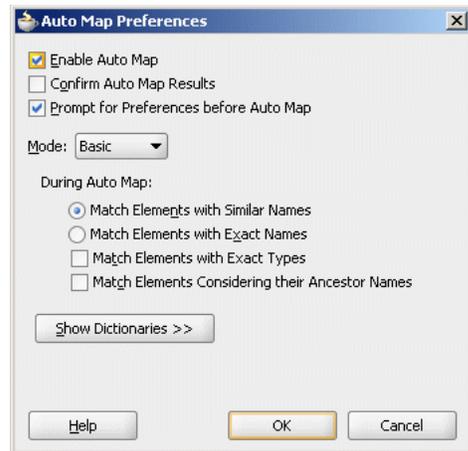
14. Drag and drop the **imp1:CustomerData** source element to **imp1:Customer** target element.

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 6–34](#).

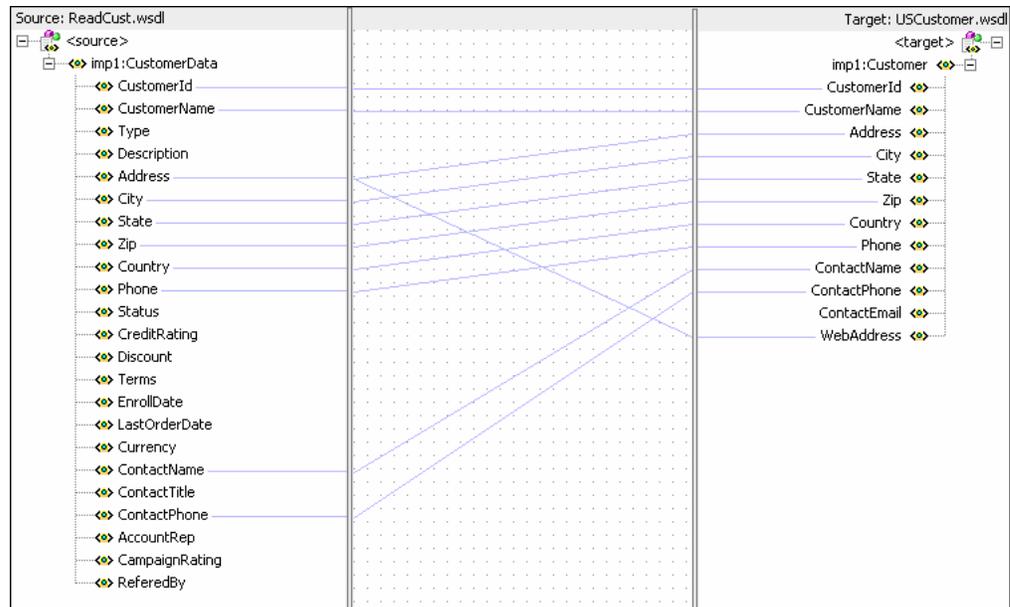
Figure 6–34 Auto Map Preferences Dialog



16. Click **OK**.

The `CustomerData_To_Customer.xsl` tab appears as shown in [Figure 6–35](#).

Figure 6–35 CustomerData_To_Customer.xsl Tab – Auto Mapped Connections



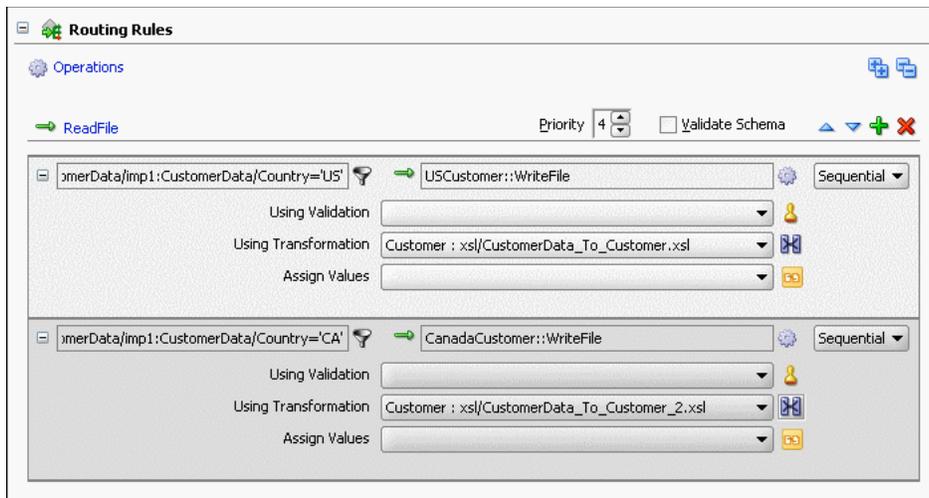
17. From the **File** menu, click **Save All**.

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:

```
$in.CustomerData/imp1:CustomerData/Country='CA'
```

[Figure 6–36](#) shows how the mediator editor would appear after you have specified `CanadaCustomer` reference as target service.

Figure 6–36 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 6–20](#).

6.3.1.6 Configuring Oracle Application Server Connection

Perform the following steps to create a connection to Oracle Application Server, which is required for deploying your SOA composite application:

1. In the Application Navigator, select Application Resources.
2. Right-click **Connections** and select **New**.
3. Select **Connections** from Categories and then select **Application Server Connection** from Items.

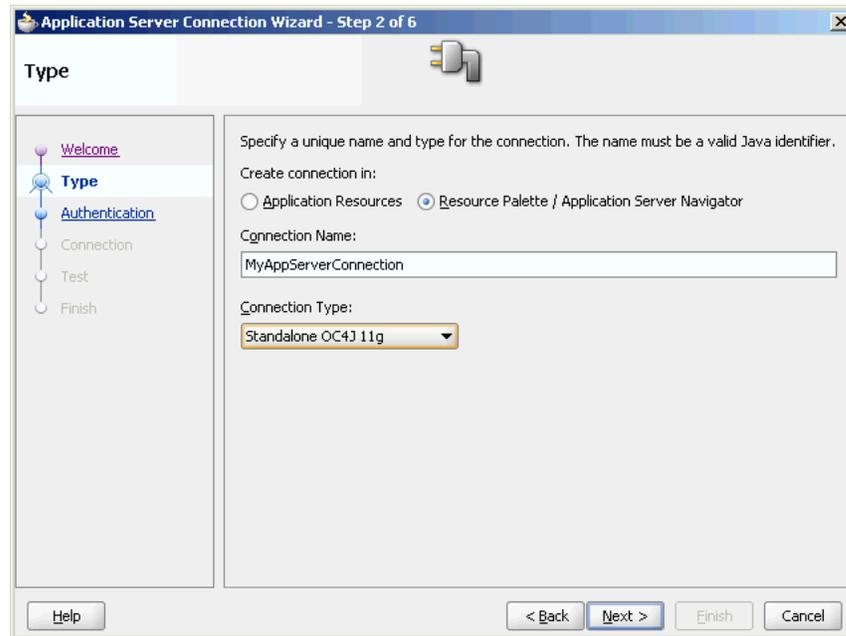
The Create Application Server Connection wizard is displayed.

4. Click **Next**.

The Create Application Server Connection Type page is displayed.

5. Enter `MyAppServerConnection` in the Connection Name field and select **Standalone OC4J 11g** from the Connection Type list, as shown in [Figure 6–37](#).

Figure 6-37 The Create Application Server Connection Type Page



6. Click Next.

The Connection Authentication page is displayed.

7. Enter the following details:

- Username: Accept the default value of **oc4jadmin**.
- Password: Enter `welcome1`.

8. Click Next. The Create Application Server Connection page is displayed.

9. Enter the local host name on which the Oracle SOA Suite infrastructure is installed.

10. Click Next.

11. Click Test Connection.

The following message should appear:

Success!

If the test is unsuccessful, ensure that Oracle Application Server is running, and retry the test.

12. Click Next.

13. Click Finish.

6.3.1.7 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 "[Deploying Applications](#)" on page 3-2.

6.3.2 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. Based on the payload, the files will be written to the specified output directories.

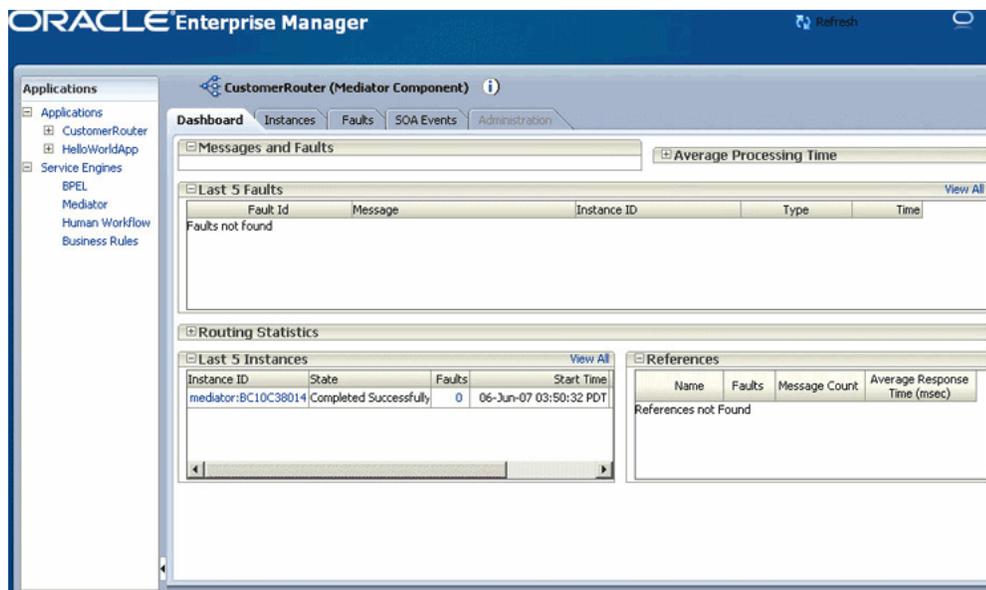
For monitoring the running instance, you can use the SOA Console at the following URL:

`http://hostname:8888/SOAConsole`

where `hostname` is the host on which you installed the Oracle SOA Suite infrastructure.

In the SOA Console, you can click the `CustomerRouterProject` to see the project dashboard as shown in [Figure 6–38](#).

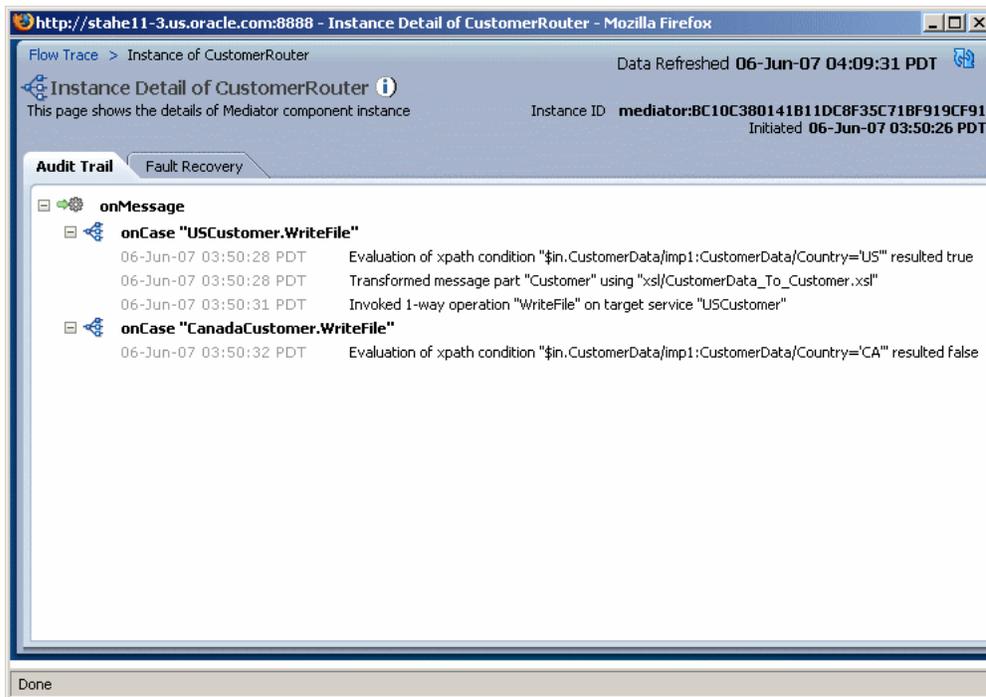
Figure 6–38 *CustomerRouterProject Dashboard in SOA Console*



You can also navigate through the Last 5 instances section to open the instance of your choice.

To view the detailed execution trail of the `CustomerRouter` mediator, click the instance id in the instance column. The Audit Trail page is displayed as shown in [Figure 6–39](#).

Figure 6–39 CustomerRouterProject Audit Trail in SOA Console



Oracle Mediator Error Handling

Oracle Mediator provides sophisticated error handling capabilities that enables you to configure an Oracle Mediator service component (mediator component) for error occurrences and corresponding corrective actions. This chapter describes how to handle errors with Oracle Mediator.

This chapter contains the following topics:

- [Introduction to Oracle Mediator Error Handling](#)
- [Using Error Handling](#)
- [XML Schema Files for Error Handling](#)

7.1 Introduction to Oracle Mediator Error Handling

Error handling enables a mediator component 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 Oracle 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. Oracle Mediator provides fault policy based error handling for system faults.

Fault policies enables you to handle errors automatically or through human intervention. Oracle Mediator fault policy based error handling consists of following two components:

- [Fault Policies](#)
- [Fault Bindings](#)

7.1.1 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 "[Fault-policies.xml Schema File](#)" on page 7-5. 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:4101">
```

```

        <condition>
            <action ref="ora-retry"/>
        </condition>
    </faultName>
</Actions>
    <Action id="ora-retry">
        <retry>
            <retryCount>3</retryCount>
            <retryInterval>2</retryInterval>
            <exponentialBackoff/>
            <retryFailureAction ref="ora-java"/>
            <retrySuccessAction ref="ora-human-intervention"/>
        </retry>
    </Action>
</Actions>
</faultPolicy>
</faultPolicies>

```

A fault policy consists of the following two components:

- [Conditions](#)
- [Actions](#)

7.1.1.1 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 schema validation is failing, you can edit the payload and resubmit the process.

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:2101">
        <condition>
            <action ref="ora-java"/>
        </condition>
    </faultName>
    <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:TYPE_FATAL_MESH">
        <condition>
            <action ref="ora-retry"/>
        </condition>
    </faultName>
    <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:3501">
        <condition>
            <action ref="ora-retry-crm-endpoint"/>
        </condition>
    </faultName>
</Conditions>

```

You can specify an action for an error type or error group while defining the conditions in a fault policy. In the previous example, `medns:2101` refers to a specific 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 errors 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 the 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 raising 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.

7.1.1.2 Actions

Actions specify the tasks that should be performed when an error occurs. Oracle Mediator provides a list of actions that you can use in a fault policy. These predefined actions are described in the following list:

- Human intervention: This action can be applied only in case of errors that occur while executing asynchronous routing rules. You can perform the following functions:
 - Retry: You can use this option to retry the process. For example, if a service is not available, then you might want to try accessing it again.
 - Change payload and resubmit: You can select this option to change the payload and resubmit it. This is useful in case of data errors such as data validation and data transformation. You can change message payload by using the SOA Console.
 - Abort the flow: You can select this option to end the process.
- Retry: This action enables you to retry the error in following ways:
 - 'N' times: Retry is performed 'N' number of times. Each time a retry is carried out, the retry count is stored in the mediator instance table to keep track of the retry count.
 - With constant delay/exponential back off: Retry handler computes the retry interval based on the configuration in the policy file. Exponential back off increases the retry interval exponentially for each retry attempt.
- Java call out: This action enables you to call a customized class that implements `oracle.tip.mediator.common.error.management.recovery` interface.
- Abort: This action enables you to abort the flow.

Fault policies can be created at the following levels:

- Composite: You can define one fault policy for all mediator components in a composite.
- Component: You can define fault policy for a mediator component exclusively. A component-level fault policy overrides the composite-level fault policy.

Human intervention is the default action for errors, which do not have a fault policy defined.

7.1.2 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 "[Fault-bindings.xml Schema File](#)" on page 7-9. Fault bindings can be done based on the fully qualified name of a composite. 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"/>
  <component dn="RetryDemo/TestResubmit!1.0*2007-04-09-23-34-36/FileInToFileOut"
    faultPolicy="CRM_ServiceFaults"/>
</faultPolicyBindings>
```

7.2 Using Error Handling

To enable the error handling for mediator components in a SOA composite, perform the following steps:

1. Create a `fault-policies.xml` file based on the schema defined in the ["Fault-policies.xml Schema File"](#) on page 7-5.
2. Create a `fault-bindings.xml` file based on the schema defined in the ["Fault-bindings.xml Schema File"](#) on page 7-9.
3. Copy the `fault-policies.xml` and the `fault-bindings.xml` file to your SOA Composite project directory.
4. Deploy the SOA Composite project.

All the fault policies for a composite are loaded when the first error occurs. At run time, 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.

7.3 XML Schema Files for Error Handling

This section describes the schema files for `fault-policies.xml` and `fault-bindings.xml` files and consists of the following topics:

- ["Fault-policies.xml Schema File"](#) on page 7-5
- ["Fault-bindings.xml Schema File"](#) on page 7-9

7.3.1 Fault-policies.xml Schema File

The `fault-policies.xml` file should be based on the following schema:

```
<?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 Recvorey 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: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 will replay 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: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>

```

7.3.2 Fault-bindings.xml Schema File

The `fault-bindings.xml` file should be based on the following schema:

```

<?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" />
                <xs:element name="component" type="tns:componentType"
minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:attribute name="version" type="xs:string" default="2.0.1" />
        </xs:complexType>
    </xs:element>

```

```
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="nameType">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="componentType">
        <xs:annotation>
            <xs:documentation>Bindings for a component. Overrides composite level
binding.</xs:documentation>
        </xs:annotation>
        <xs:attribute name="dn" type="tns:nameType" use="required"/>
        <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>
```

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 8.1, "Introduction to Business Events"](#)
- [Section 8.2, "Creating Business Events in Oracle JDeveloper"](#)

8.1 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

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 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 — 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 8-1](#) 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 8-1 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. Once an EDL is published, it can be subscribed to by other applications. EDLs cannot be unpublished; the definition always exists.

There are three levels of event subscription:

- All events with 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` in conjunction with a namespace declaration such as `xmlns:prefix=URI` or a namespace context.
- All events within a given namespace (for example, `x.y.z/*`)
- All events broadcast on the EDN

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.
- Complex event processing (CEP) can subscribe to and publish events.

Notes:

- BPEL processes cannot directly publish events. However, you can use a BPEL process to invoke Oracle Mediator and have Oracle Mediator send the event.
 - There are two implementations of the EDN: JMS and AQ (provides support for PL/SQL APIs).
-
-

8.1.1 Local and Remote Events Boundaries

A single SOA composite application instance can reside in a single OC4J container or can be clustered across multiple OC4J containers. Another application (for example, an ADF BC application) can be configured to run in the same OC4J container as the SOA composite application instance or in a different container.

Raising an event outside of an SOA composite application instance can be done through a local event connection or a remote event connection.

- Local event connection — If the caller resides in the same OC4J container as the application and the caller 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 OC4J container (different JVM) as the application, then the event is raised through a remote event connection. Only asynchronous subscriptions are supported for remote event connections.

If another application (for example, an ADF BC application) is configured to run in the same OC4J 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 a mediator in both instances, which listens to events and publishes them to a JMS queue.

8.1.2 Synchronous Subscriptions

Table 8–1 provides a summary of the event publishing and subscription combinations that support synchronous subscriptions. For synchronous subscriptions, the subscriber’s logic is executed in the same transaction as the publisher.

Table 8–1 Synchronous Subscriptions

Same OC4J Container for Publisher and Subscriber?	Publisher	Subscriber	Synchronous Subscription Executed Synchronously (Local Event Connection)?
No (configurable)	ADF BC event object through create, retrieve, update, delete) CRUD or through the Event API	ESB routing rule	No

Table 8–1 (Cont.) Synchronous Subscriptions

Same OC4J Container for Publisher and Subscriber?	Publisher	Subscriber	Synchronous Subscription Executed Synchronously (Local Event Connection)?
No (configurable)	Same as above	PL/SQL invoke Event API	No
No (configurable)	Same as above	Any Java invoke Event API	No
Yes (configurable)	ADF BC event object through CRUD or through Event API	ESB routing rule	Yes
Yes (configurable)	Same as above	PL/SQL invoke Event API	Yes
Yes (configurable)	Same as above	Any Java invoke Event API	Yes
Yes	PL/SQL invoke Event API	ESB routing rule	No
Yes	Same as above	PL/SQL invoke Event API	Yes
Yes	Same as above	Any Java invoke Event API	Yes
Yes	Mediator	ESB routing rule	Yes
Yes	Same as above	PL/SQL invoke a Event API	Yes
Yes	Same as above	Any Java invoke Event API	Yes
No. The publisher and subscriber are in different application instances.	All	All	Not applicable. Subscriptions across application instances are not supported. Implement the bridge mechanism to propagate event.

8.2 Creating Business Events in Oracle JDeveloper

This section provides a high-level overview of creating a composite application in which a Oracle Mediator service component subscribes to a business event and invokes a BPEL process.

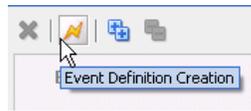
Note: If you want to use a business event with a BPEL process, you must use Oracle Mediator to subscribe to the event. This is because BPEL can only publish, and not subscribe, to events.

8.2.1 How to Create a Business Event

1. Create a SOA project as an empty composite.
2. Start the Event Definition Creation wizard in either of two ways:

- a. Click the icon above the canvas workspace in the SOA Composite Editor. [Figure 8–1](#) provides an example.

Figure 8–1 Event Definition Creation



- b. Select **New > SOA Tier > Service Components > Event Definition** from the **File** main menu.

The Event Definition Creation wizard appears.

3. Enter the following details.

Field	Value
Name	Enter a name.
Directory	Accept the default directory path or enter a specific value.
Namespace	Accept the default namespace or enter a specific value for the namespace in which to place the event.

4. Click **Finish**.

The Events editor appears. The events you define are saved in the *event_definition_name.edl* file.

5. Click the + sign to add an event.

The Add an Event window appears.

6. Enter the following details.

Field	Value
Element	Click the flashlight icon to select the payload (typically an XSD file).
Event	Enter a name.

7. Click **OK**.

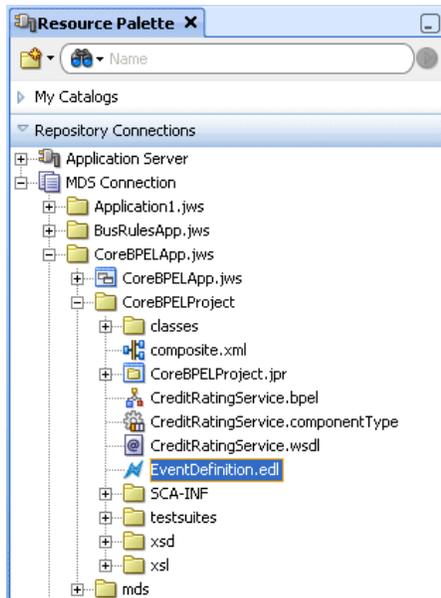
The added event now appears in the **Events** section.

8. 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. [Figure 8–2](#) shows that the event displays for browsing in the **Resource Palette** in Oracle JDeveloper.

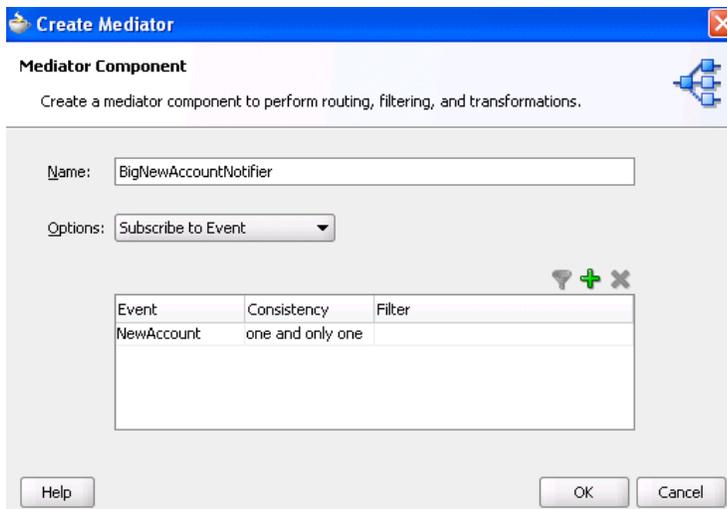
Figure 8–2 Business Event in the Resource Palette



8.2.2 How to Subscribe to a Business Event

1. Drag and drop a **Mediator** service component into the SOA Composite Editor. This service component enables you to subscribe to the business event.
2. Enter a name in the **Name** field.
3. Select **Subscribe to Event** from the **Options** list.
The window is refreshed to display an events table.
4. Click the + sign 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 shown in [Figure 8–3](#).

Figure 8–3 Create Mediator Dialog



6. Select a level of delivery consistency for the event.
 - **one and only one** — events are delivered to the subscriber in the context of its own global (that is, JTA) transaction. Any changes made by the subscriber within the context of that transaction are committed once the event processing is complete. If the subscriber fails, the transaction is rolled back. Failed events are retried a configured number of times before being delivered to the hospital queue.
 - **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 on the same 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.
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 window. This enables you to specify an XPath expression. When the expression logic is satisfied, the event is accepted for delivery.
8. Click **OK**.

Figure 8–4 shows an icon on the left side that indicates that Mediator is configured for an event subscription.

Figure 8–4 Configuration for Event Subscription



9. Click **Source**.

The source code in Example 8–2 provides details about the subscribed event of the Oracle Mediator service component.

Example 8–2 Subscribed Event

```
<component name="BigNewAccountNotifier">
  <implementation.mediator src="BigNewAccountNotifier.mplan"/>
  <business-events>
    <subscribe xmlns:sub1="http://xmlns.oracle.com/Project1/EventDefinition1"
              name="sub1:NewAccount" consistency="oneAndOnlyOne"/>
  </subscribe>
  </business-events>
</component>
```

While not explicitly demonstrated in this example, you can define XPath filters on events. In [Example 8-3](#), the event is accepted for delivery *only* if the initial deposit is greater than 50000:

Example 8-3 Definition of XPath Filters on Events

```
<component name="BigNewAccountNotifier">
  <implementation.mediator src="BigNewAccountNotifier.mplan"/>
  <business-events>
    <subscribe xmlns:sub1="http://xmlns.oracle.com/Project1/EventDefinition1"
      name="sub1:NewAccount" consistency="oneAndOnlyOne"/>
    <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>
    </subscribe>
  </business-events>
```

You can also subscribe to an entire namespace of events. In [Example 8-4](#), all events in the namespace `http://bigbank.com/events` are available for subscription.

Example 8-4 Subscription to an Entire Namespace of Events

```
<component name="BigNewAccountNotifier">
  <implementation.mediator src="BigNewAccountNotifier.mplan"/>
  <business-events>
    <subscribeNamespace namespace="http://bigbank.com/events"
      consistency="oneAndOnlyOne"/>
    </subscribe>
  </business-events>
</component>
</composite>
```

8.2.3 How to Publish a Business Event

You can create a second mediator to publish the event that you subscribed to in [Section 8.2.2, "How to Subscribe to a Business Event"](#) on page 8-6.

1. Create a second mediator service component that publishes the event to which the first mediator subscribes.
2. Return to the first mediator service component.
3. Click the + sign in the **Routing Rules** section.
4. Click **Service** when prompted by the Target Type window.
5. Select the second mediator service component.
6. Select **Save All** from the **File** main menu.
7. Click **Source** for the **composite.xml** file.

Note that the two mediator service components appear in [Example 8-5](#). One service component (`BigNewAccountNotifier`) subscribes to the event and the other service component (`PublishBigNewAccountNotifier`) publishes the event.

Example 8–5 Event Subscription and Publication

```

<component name="PublishBigNewAccountNotifier">
  <implementation.mediator src="PublishBigNewAccountNotifier.mplan"/>
  <business-events>
    <publishes xmlns:pub1="http://xmlns.oracle.com/Project1/EventDefinition1"
name="pub1:NewAccount"/>
  </business-events>
</component>

<component name="BigNewAccountNotifier">
  <implementation.mediator src="BigNewAccountNotifier.mplan"/>
  <business-events>
    <subscribe xmlns:sub1="http://xmlns.oracle.com/Project1/EventDefinition1"
name="pub1:NewAccount" consistency="oneAndOnlyOne">
  </subscribe>
</business-events>
</component>

```

If you define an XPath filter on the subscribed event, the syntax can look as shown in [Example 8–6](#):

Example 8–6 Definition of XPath Filter on the Subscribed Event

```

<component name="PublishBigNewAccountNotifier">
  <implementation.mediator src="PublishBigNewAccountNotifier.mplan"/>
  <business-events>
    <publishes xmlns:pub1="http://xmlns.oracle.com/Project1/EventDefinition1"
name="pub1:NewAccount"/>
  </business-events>
</component>

<component name="BigNewAccountNotifier">
  <implementation.mediator src="BigNewAccountNotifier.mplan"/>
  <business-events>
    <subscribe xmlns:sub1="http://xmlns.oracle.com/Project1/EventDefinition1"
name="sub1:NewAccount" consistency="oneAndOnlyOne"/>
  <filter>
    <xpath xmlns:be="http://oracle.com/fabric/businessEvent"
xmlns:ns1="http://xmlns.oracle.com/singleString"
  <xpath expression=" /be:business-event/be:content/
pub1:AccountInfo/Details[@initialDeposit > 50000]" />
  </filter>
</subscribe>
</business-events>

```

8.2.4 How Integrate ADF BC Business Events with Oracle Mediator

While not shown in this example, you can also create Application Development Framework (ADF) business component (BCs) event conditions to which you subscribe in Oracle Mediator.

This section provides a high-level overview of the steps to follow.

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 ADF BC upon creation.
3. Create a SOA composite application and manually include copies of the EDL and XSD files in the project's directory path.

4. Create a mediator service component as described in [Section 8.2.2, "How to Subscribe to a Business Event"](#) on page 8-6.
5. Select the event's EDL file in the Event Chooser window as described in [Section 8.2.2, "How to Subscribe to a Business Event"](#) on page 8-6.
6. Create a BPEL process service component in the same SOA composite application for the mediator to invoke. In the **Input Element** field of the **Advanced** tab, ensure that you select the payload of the BC business event XSD created in Step 2.
7. Double-click the BPEL process.
8. Drag and drop an **Email** activity into the BPEL process.
9. Use the payload of the business event XSD to complete the **Subject** and **Body** fields.
10. Return to the mediator service component in the SOA Composite Editor.
11. Design a second service component to publish the event, such as a BPEL process or a second mediator service component.

SOA composite application design is now complete.

For more information about creating ADF BC business events, see *Oracle Fusion Applications Developer Standards and Guidelines*.

Working with Domain Value Maps

This chapter describes how to use domain value maps to enable mapping of vocabulary in one domain to another.

This chapter includes the following topics:

- [Introduction to Domain Value Maps](#)
- [Creating Domain Value Maps](#)
- [Using Domain Value Map Functions](#)
- [Domain Value Map Use Case](#)

9.1 Introduction to Domain Value Maps

Many a times, applications that you want to integrate use different vocabulary to represent the same information. For example, one domain might represent a city with the long name (Boston) while another domain may represent a city with a short name (BO). In such cases, you can use a domain value map to directly map values between multiple domains. A direct mapping of values between two or more domains is also known as point-to-point mapping. [Table 9–1](#) shows a point-to-point mapping for cities between two domains:

Table 9–1 *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.

9.1.1 Domain Value Map Features

The domain value map functionality consists of the following features:

- [Qualifier Support](#)

- [Qualifier Order Support](#)
- [One-to-Many Mapping Support](#)

9.1.1.1 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 as Kensington is a city in Canada as well as USA. Hence this mapping requires a qualifier (USA or Canada) to qualify when the mapping becomes valid as shown in [Table 9-2](#).

Table 9-2 *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 9-3](#), BELG to Belgrade mapping can also be qualified with state name.

Table 9-3 *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. Therefore, the qualifier values can not be looked up.

9.1.1.2 Qualifier Order Support

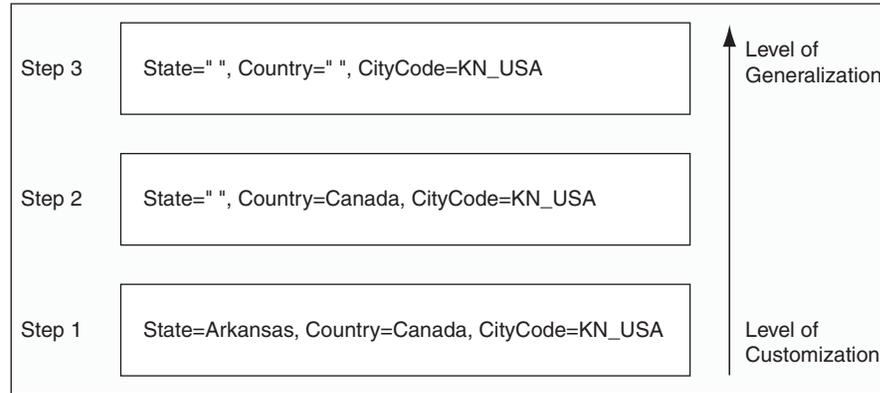
A qualifier order is used to find the best match during lookup at run time. The order of a qualifier varies from highest to lowest depending on the role of the qualifier in defining a more exact match. In [Table 9-3](#), the state qualifier can have a higher order than the country qualifier, as a matching 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 " ". It proceeds until a match is found, or until a match is not found with all qualifiers set to " ". [Figure 9-1](#) describes hierarchical lookup performed for the following lookup on [Table 9-3](#):

State=Arkansas, Country=Canada, CityCode=KN_USA

In this example, the `State` qualifier has a qualifier value as 2 and the `Country` qualifier has a qualifier value as 1.

Figure 9–1 Hierarchical Lookup Example



As shown in [Figure 9–1](#), 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 9–4](#) provides a summary of these steps.

Table 9–4 Domain Value Map Lookup Result

STATE	COUNTRY	ShortValue	Lookup Result
Arkansas	CANADA " "	KN_USA	No Match
" "	CANADA	KN_USA	No Match
" "	" "	KN_USA	Kensington

9.1.1.3 One-to-Many Mapping Support

You can map one value to a 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 9–5](#).

Table 9–5 One-to-Many Mapping Support

Discount Percentage	Discount Period	NetCredit Period	Payment Term
10	20	30	GoldCustomerPaymentTerm
5	20	30	SilverCustomerPaymentTerm
2	20	30	RegularPaymentTerm

9.2 Creating Domain Value Maps

You can create one or more domain value maps, in a SOA Composite application of Oracle JDeveloper and then at run time, use it to look up for column values.

9.2.1 How to Create Domain Value Maps

TBD

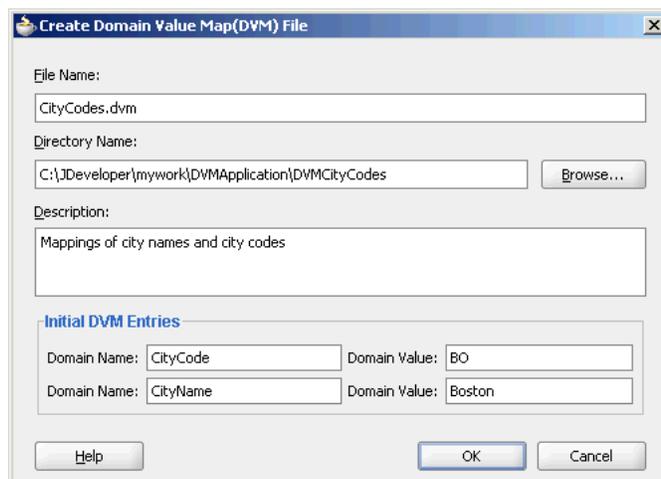
To create a domain value map:

1. In the Application Navigator, right-click the project in which you want to create a domain value map and select **New**.
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** to open the Create Domain Value Map(DVM) File dialog.
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 of 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 9–2](#).

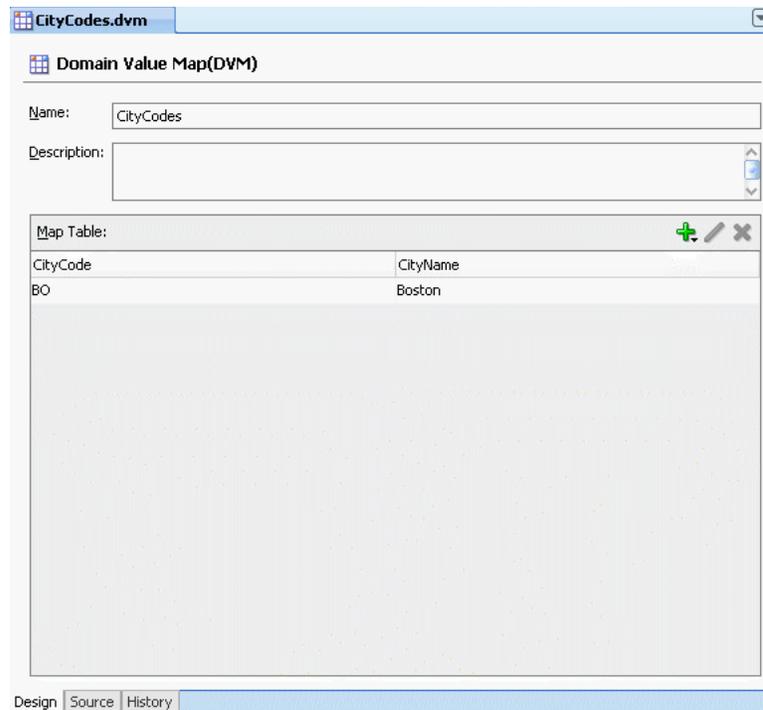
Figure 9–2 Populated Create Domain Value Map File Dialog



8. Click **OK**.

The domain value map editor is displayed, as shown in [Figure 9–3](#).

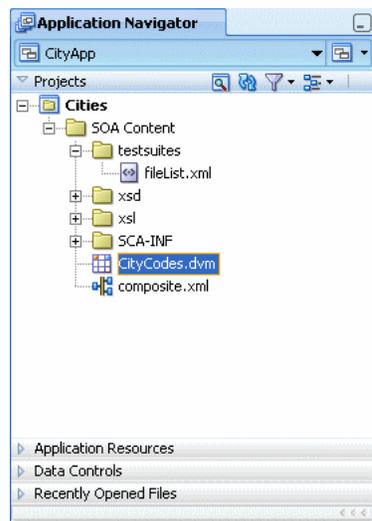
Figure 9–3 Domain Value Map Editor



9.2.2 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 9–4](#).

Figure 9–4 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:string"
use="optional"/>
                <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>

```

9.3 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.

9.3.1 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. Select **True** to set this column as a Qualifier.
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**.

9.3.2 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**.

9.3.3 Reordering the Columns in a Domain Value Map

You can move a column from one position to another. This feature is provided to support user preferences, it has no effect on how the domain value map is used at run time.

To reorder a column in a domain value map:

1. Select the column that you want to move.
2. Drag the column to move the selected column.
3. Repeat steps 1 and 2 until all the columns appear in the desired order.

9.4 Using Domain Value Map Functions

After creating a domain value map, you can use the domain value maps XPath functions to look up for appropriate values and populate the targets for the applications at run time.

9.4.1 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 or multiple values at run time.

9.4.1.1 `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('C:\dvms\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('C:\dvms\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.

9.4.1.2 `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 DocumentFragment
```

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  
( 'C:\dvms\cityMap.dvm', 'CityCodes', 'BO', 'CityShortNames', 'CityAbbreNames' )
```

9.4.2 Using Domain Value Map Functions in Transformation

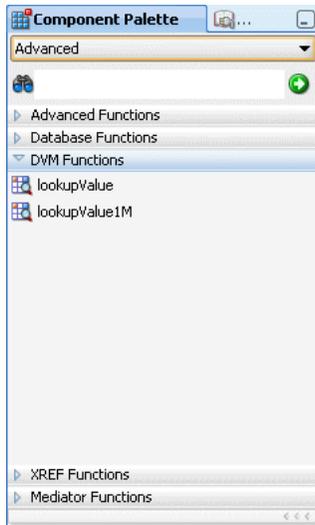
The domain value map functions can be used for transformation with a BPEL service component or a Mediator service component. Transformations are done by using the XSLT Mapper window, which is displayed when you create an XSL file to transform the data from one XML schema to another.

See Also: [Chapter 4, "XSLT Mapper and Transformations"](#) for information about XSLT Mapper.

To use the `lookupValue1M` Function in Transformation:

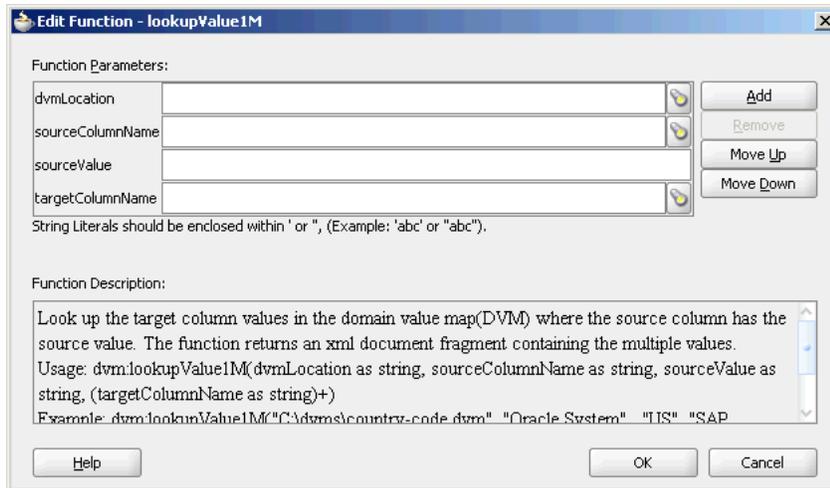
1. In the Application Navigator, double-click an XSL file to open the XSLT Mapper window.
2. In the XSLT Mapper window, expand the trees in the Source and Target panes.
3. In the Component Palette, click down arrow and then select **Advanced**.
4. Select **DVM Functions** as shown in [Figure 9-5](#).

Figure 9–5 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 9–6](#).

Figure 9–6 Edit Function – lookupValue1M Dialog



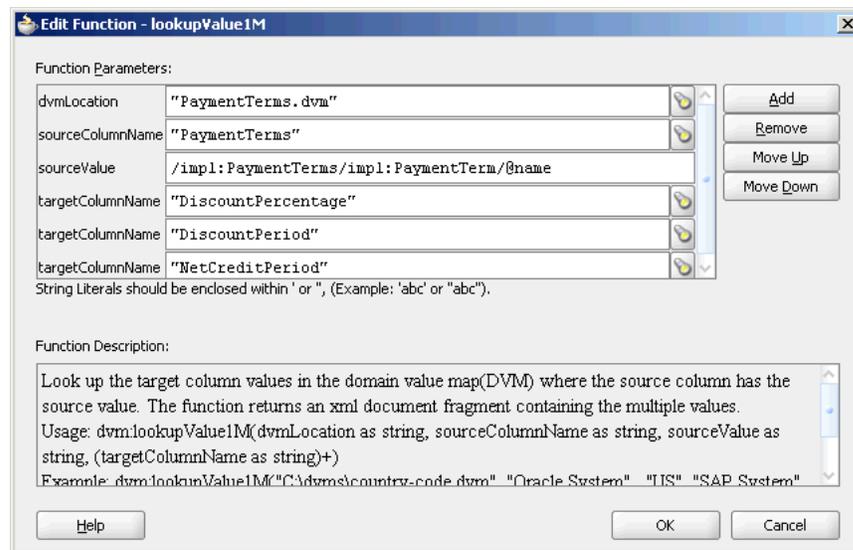
7. Specify values for the following fields in the Edit Function – lookupValue1M dialog:
 - a. In the **dvmLocation** field, enter the location URI of the domain value map file or click **Browse** to the right of the `dvmLocation` field to select a domain value map file. You can select an already deployed domain value map from MDS and also from 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 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 9-7](#).

Figure 9-7 Populated Edit Function – lookupValue1M Dialog



8. Click **OK**.

The XSLT mapper window is displayed with lookupValue1M function icon.

9. From the **File** menu, click **Save All**.

9.4.3 Using a Domain Value Map Functions to Create 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 Assign activity of a BPEL service component or through Assign Values functionality of a Mediator service component.

Note: The lookupValue1M function is currently not supported in the Expression Builder dialog.

See Also: [Section 6.2.1.6, "Assigning Values"](#) for information about Assign Values functionality.

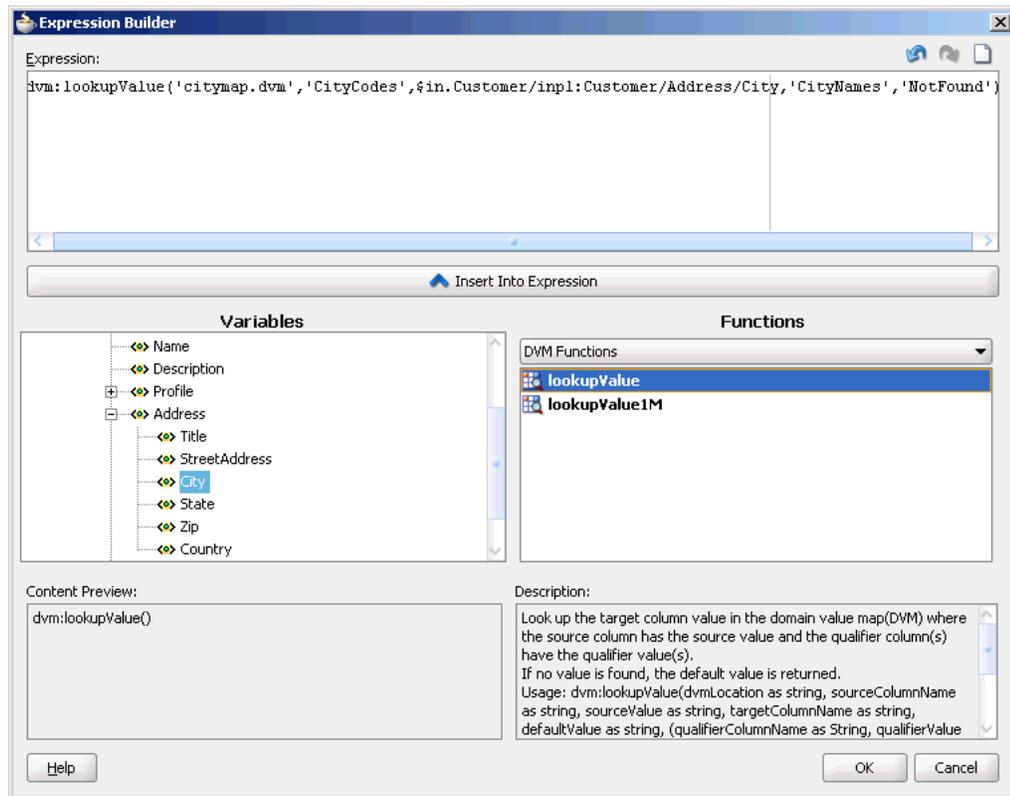
To use the lookupValue function in the Expression Builder dialog:

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 9–8](#), looks up a domain value map for city name equivalent of a city code. The value of the city code depends upon the value specified at the run time.

Figure 9–8 Domain Value Map Functions in Expression Builder Dialog



9.4.4 What Happens at Run Time

At run time, a BPEL service component or a Mediator service component uses the domain value map to look up appropriate values and populate the targets for the applications which a BPEL service component or a mediator service component is integrating.

9.5 Domain Value Map Use Case

This sample demonstrates the hierarchical lookup feature of domain value map. 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 `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.

9.5.1 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.

- [Task 1: Creating an Oracle JDeveloper Application and Project](#)
- [Task 2: Creating a Domain Value Map](#)
- [Task 3: Creating a File Adapter Service](#)
- [Task 4: Creating ProcessOrders Mediator](#)
- [Task 5: Creating a File Adapter Reference](#)
- [Task 6: Specifying Routing Rules](#)
- [Task 7: Configuring Oracle Application Server Connection](#)
- [Task 8: Deploying the Composite Application](#)

9.5.1.1 Task 1: Creating an Oracle JDeveloper Application and Project

To create an application and a project for the use case:

1. In Oracle JDeveloper, select the **Applications Navigator** tab.
2. In the Application Navigator, right-click and select **New**.
3. In the New Gallery, expand the **General** node, and select the **Applications** category.
4. In the **Items** list, select **Application** and click **OK** to open the Create Application dialog.
5. In the **Application Name** field, enter `Hierarchical`, and then click **OK** to open the Create Project dialog.
6. In the **Project Name** field, enter `HierarchicalValue` and click **OK**.
7. In the Application Navigator, right-click the **HierarchicalValue** project and select **New**.
8. In the New Gallery, expand the **SOA Tier** node.
9. In the **Items** list, select **SOA Composite** and click **OK** to open the Create SOA Composite dialog.
10. In the Composite Template list, select **Empty Composite** and then click **OK**.

The Applications Navigator of Oracle JDeveloper is updated with the new application and project and the Design tab contains, a blank palette.

11. From the **File** menu, click **Save All**.

9.5.1.2 Task 2: Creating a Domain Value Map

After creating an application and a project for the use case, you need to 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** to open the Create Domain Value Map(DVM) File dialog.
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** to open the domain value map editor.
9. Click **Add** and then select **Add Column** to open the Create DVM Column dialog.
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 9-9](#).

Figure 9–9 UnitsOfMeasure Domain Value Map

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.

9.5.1.3 Task 3: Creating a File Adapter Service

After creating the the domain value map, you need to create a File adapter service, named `ReadOrders` to read the XML files from a directory.

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**.

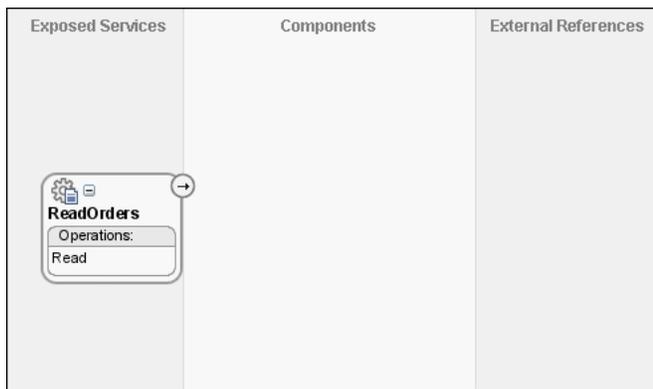
The Finish page is displayed.

17. Click **Finish**.

18. From the **File** menu, click **Save All**.

Figure 9–10 shows the `ReadOrders` service in SOA Composite Editor.

Figure 9–10 *ReadOrders Service in the SOA Composite Editor*



9.5.1.4 Task 4: Creating ProcessOrders Mediator

Perform the following steps to create a Mediator named `ProcessOrders`:

1. Drag and drop a Mediator from Components 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**.

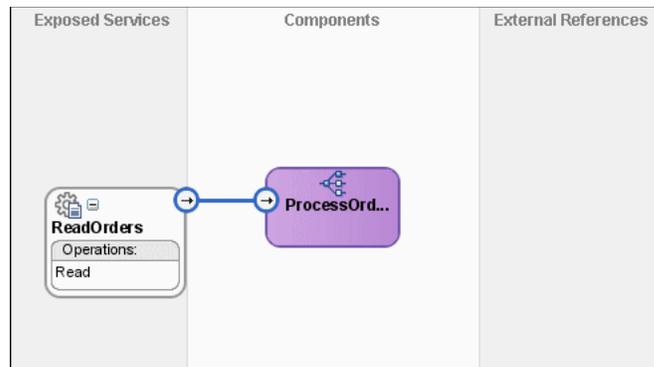
4. Click **OK**.

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 9–11.

This specifies the file adapter service to invoke the `ProcessOrders` mediator while reading a file from the input directory.

Figure 9–11 ReadOrders Service Connected to the ProcessOrders Mediator



6. From the **File** menu, click **Save All**.

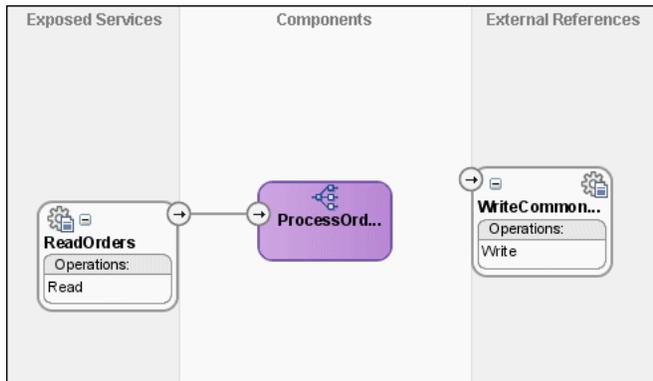
9.5.1.5 Task 5: Creating a File Adapter Reference

Perform the following steps to create a file adapter reference, named `WriteCommonOrder`:

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 9–12 shows the `WriteCommonOrder` reference in SOA Composite Editor.

Figure 9–12 *WriteCommonOrder Reference in SOA Composite Editor*



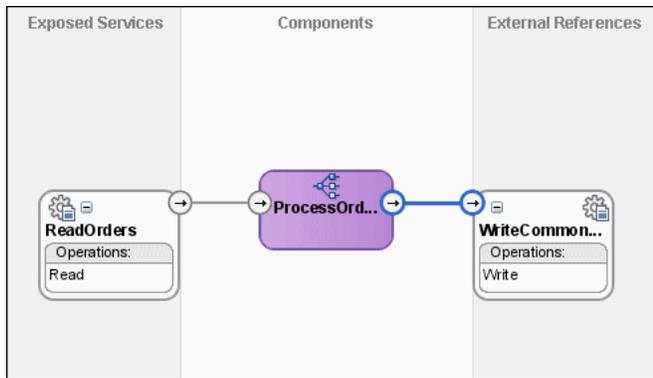
15. From the **File** menu, click **Save All**.

9.5.1.6 Task 6: Specifying Routing Rules

Follow these steps to specify the path that messages takes from the `ReadOrders` adapter service to the external reference:

1. Connect the **ProcessOrders** mediator to the **WriteCommonOrder** reference as shown in Figure 9–13.

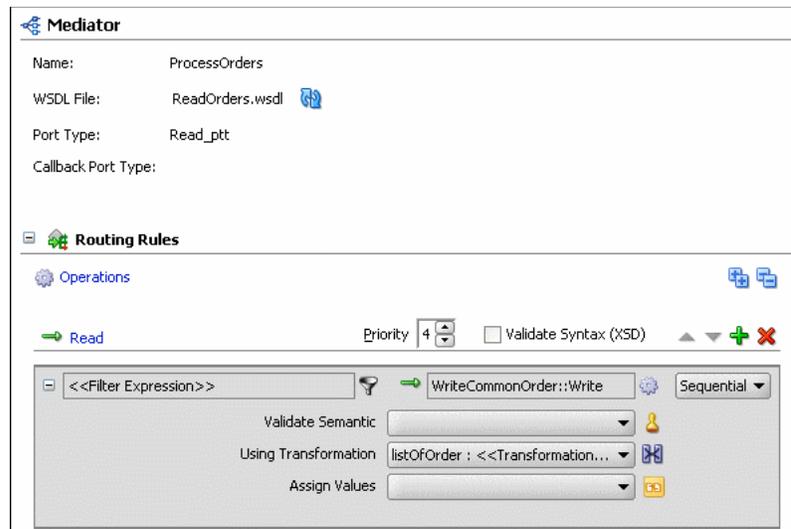
Figure 9–13 *ProcessOrders Mediator Connected to the WriteCommonOrder Reference*



2. Double-click **ProcessOrders** mediator.

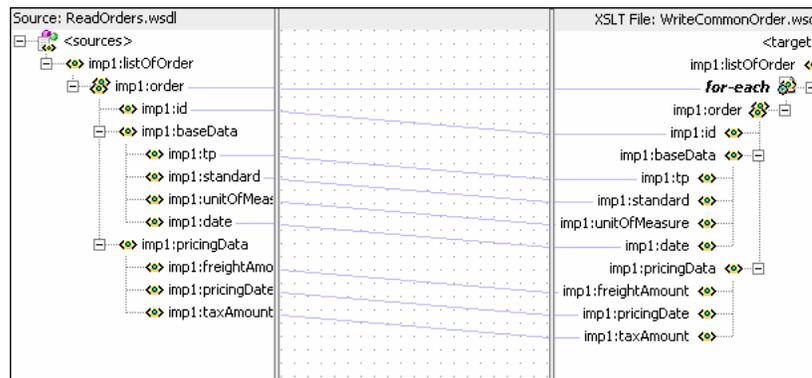
The Mediator Editor is displayed, as shown in Figure 9–14.

Figure 9–14 ProcessOrders Mediator in Mediator Editor



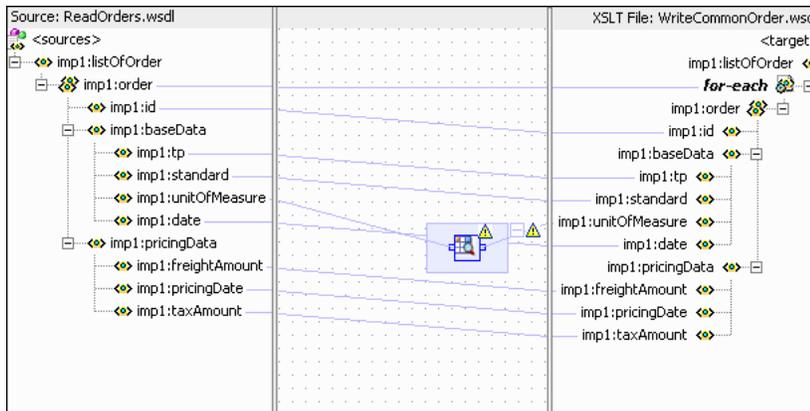
3. Click the icon to the right of the **Using Transformation** field.
The Request Transformation Map dialog is displayed.
4. Select **Create New Mapper File** and click **OK**.
A listOfOrder_To_listOfOrder.xsl tab is displayed.
5. Drag and drop the **imp1:listOfOrder** source element to **imp1:listOfOrder** 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 9–15](#).

Figure 9–15 imp1:listOfOrder To imp1:listOfOrder Transformation



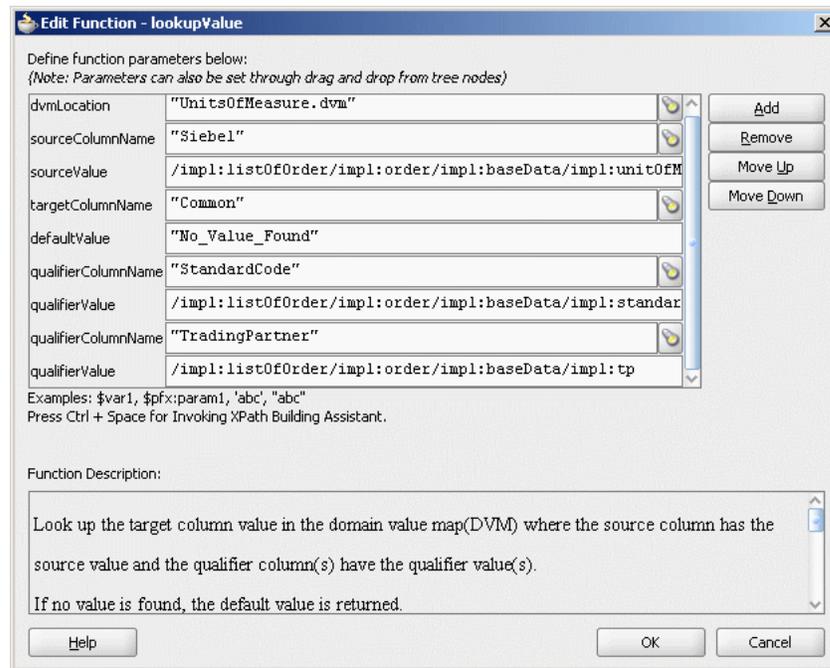
8. In the Components Palette, select **Advanced**.
9. Click **DVM Functions**.
10. Drag and drop **lookupValue** on the line connecting the unitsOfMeasure elements, as shown in [Figure 9–16](#).

Figure 9–16 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 `dvmLocation` field.
The SCA Resource Lookup dialog is displayed.
13. Select **UnitsofMeasure.dvm** and click **OK**.
14. Click **Search** to the right of `sourceColumnName` field.
The Select DVM Column dialog is displayed.
15. Select **Siebel** and click **OK**.
16. In the **sourceValue** column, enter
`/imp1:listOfOrder/imp1:order/imp1:baseData/imp1:unitOfMeasure`
 .
17. Click **Search** to the right of `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
`/imp1:listOfOrder/imp1:order/imp1:baseData/imp1: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
`/imp1:listOfOrder/imp1:order/imp1:baseData/imp1:tp`.
 The Edit Function-lookupValue dialog would appear as shown in [Figure 9–17](#).

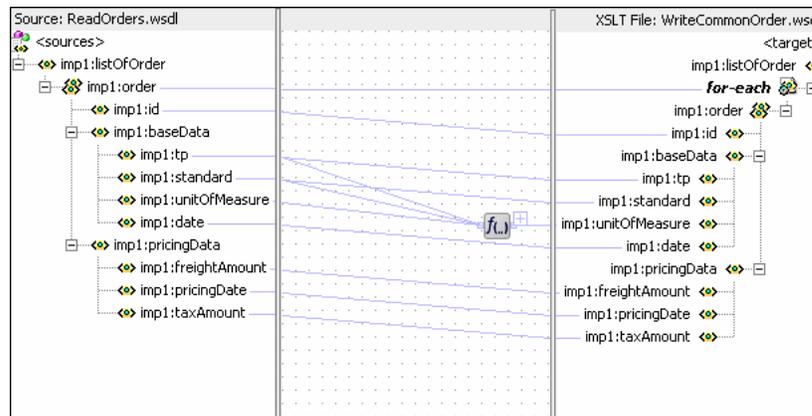
Figure 9–17 Edit Function-lookupValue Function Dialog: Hierarchical Lookup Use Case



28. Click **OK**.

The Transformation would appear as shown in [Figure 9–18](#).

Figure 9–18 Complete `imp1:listOfOrder` To `imp1:listOfOrder` Transformation



29. From the **File** menu, click **Save All** and close the `listOfOrder_To_listOfOrder.xml` tab.

9.5.1.7 Task 7: Configuring Oracle Application Server Connection

An Oracle Application Server connection is required for deploying your SOA composite application. For information on creating Oracle Application Server connection, refer to [Section 9.5.1.7, "Task 7: Configuring Oracle Application Server Connection"](#).

9.5.1.8 Task 8: Deploying the Composite Application

Deploying the `HierarchicalValue` 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 3.4, "Deploying Applications".

9.5.2 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.

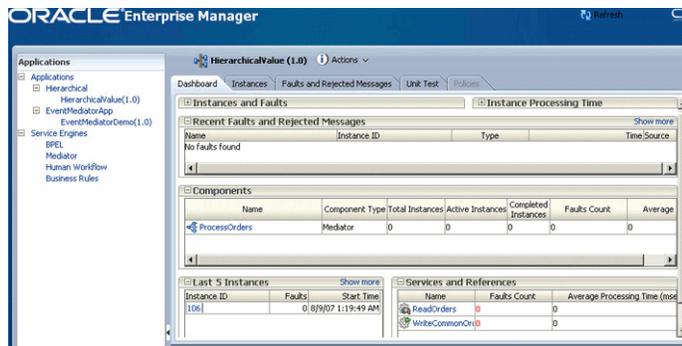
For monitoring the running instance, you can use Oracle Enterprise Manager Console at the following URL:

`http://hostname:port/em`

where `hostname` is the host on which you installed the Oracle SOA Suite infrastructure.

In Oracle Enterprise Manager Console, you can click the `HierarchicalValue` to see the project dashboard, as shown in [Figure 9–19](#).

Figure 9–19 *HierarchicalValue Dashboard in Oracle Enterprise Manager Grid Control Console*



To view the detailed execution trail, click the instance id in the instance column. The Flow Trace page is displayed as shown in [Figure 9–20](#).

Figure 9–20 HierarchicalValue Flow Trace in Oracle Enterprise Manager Grid Control Console

Flow Trace Data Refreshed 09-Aug-07 01:20:25 PDT
ECID JCA:37d46b22-d4ca-4803-9253-e6743c5e571a:1186647589195

This page shows the flow of the message through various composite and component instances

Faults
Click on a fault message to locate that fault in the trace view below

Message	Type	Time	At	Composite Instance	Application
No Faults Found					

Trace
Click on a component instance to see its detailed audit trail

Show Instance

Instance	Type	Status	Time	Composite Instance	Application
ReadOrders	Service	Completed Successfully	09-Aug-07 01:19:49 PDT	HierarchicalValue of 106	Hierarchical
ProcessOrders	Mediator Component	Completed Successfully	09-Aug-07 01:19:50 PDT	HierarchicalValue of 106	Hierarchical
WriteCommonOrder	Reference	Completed Successfully	09-Aug-07 01:19:49 PDT	HierarchicalValue of 106	Hierarchical

Working with Cross References

The cross referencing feature of Oracle SOA Suite enables you to associate identifiers for equivalent entities created in different applications. For example, you can use cross references to associate a customer entity created in one application (with native id `Cust_100`) with an entity for the same customer in another application (with native id `CT_001`).

This chapter explains how to create, populate, and use cross references. It contains the following topics:

- [Introduction to Cross References](#)
- [Creating and Modifying Cross Reference Tables](#)
- [Populating Cross Reference Tables](#)
- [Looking Up Cross Reference Tables](#)
- [Deleting a Cross Reference Table Value](#)
- [Schema Definition\(XSD\) File for Cross References](#)
- [Cross Reference Use Case](#)

10.1 Introduction to Cross References

Many a times, when you create or update objects in one application, you 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 a 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, a 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 cross references tables. [Table 10–1](#) shows a cross reference table containing information about customer identifiers in different applications.

Table 10–1 *Cross Reference Table Sample*

SAP	EBS	SBL
SAP_001	EBS_1001	SBL001

Table 10–1 (Continued) Cross Reference Table Sample

SAP	EBS	SBL
SAP_002	EBS_1002	SBL002

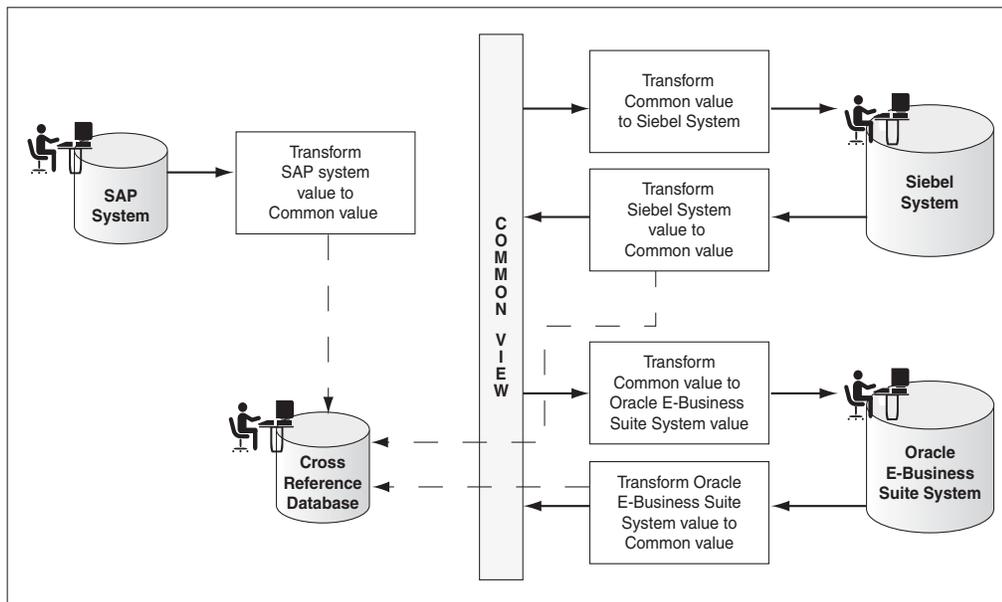
The identifier mapping is also required when information about a customer is updated in one application and the changes need to be propagated in other applications also. You can also integrate different identifiers is by using a common value integration pattern, which maps to all identifiers in a cross reference table. For example, you can add one more column *Common* to the cross reference table shown in Table 10–1. The updated cross reference table would appear, as shown in Table 10–2.

Table 10–2 Cross Reference Table with Common Column

SAP	EBS	SBL	Common
SAP_001	EBS_1001	SBL001	CM001
SAP_002	EBS_1002	SBL002	CM002

Figure 10–1 shows how you can use common value integration pattern to map identifiers in different applications.

Figure 10–1 Common Value Integration Pattern Example

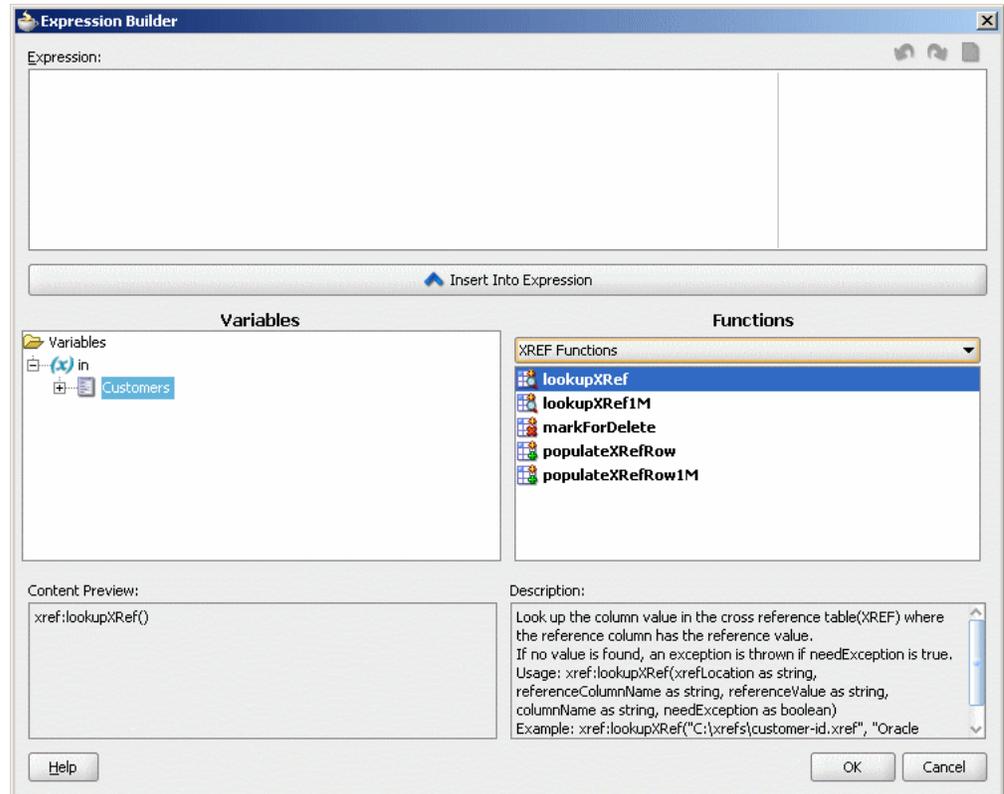


A cross reference table consists of following two parts: metadata and the actual data. The metadata is the `.xref` file created in Oracle JDeveloper, and is stored in Metadata Services (MDS) as an xml file. The actual data is stored in the database.

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 run time. However, before using a cross reference to lookup a particular value, you need to populate it at run time. 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 dialog to create transformations.

You can access the Expression builder dialog through Assign activity of a BPEL service component or through Assign value functionality of a Mediator service component. [Figure 10–2](#) shows how you can select the cross reference functions in the Expression builder dialog.

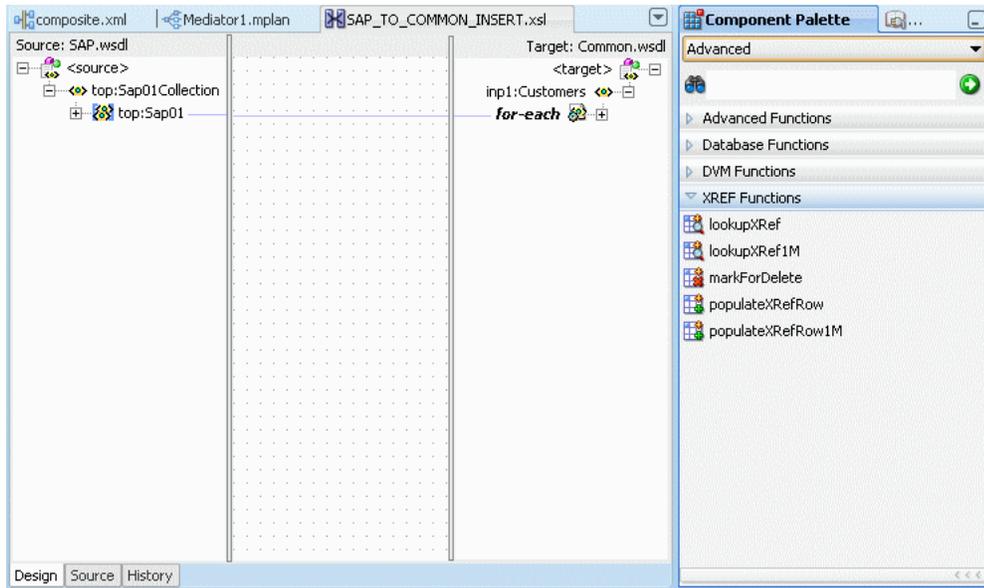
Figure 10–2 Expression Builder Dialog with Cross Reference Functions



Note: The `lookupXRef1M` function is currently not supported in the Expression Builder dialog.

The XSLT Mapper dialog is displayed when you create an XSL file to transform data from one XML schema to another. [Figure 10–3](#) shows how you can select the cross reference functions in the XSLT Mapper dialog.

Figure 10–3 XSLT Mapper Dialog with Cross Reference Functions



10.2 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 service component during transformations. The following sections explain how to create, modify, and delete cross references:

- [Creating a Cross Reference Table](#)
- [Adding a Column to a Cross Reference Table](#)
- [Deleting a Column from a Cross Reference Table](#)

10.2.1 Creating a Cross Reference Table

Perform the following steps 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. Specify the name of the cross reference file in the **File Name** field. For example, specify *Customer*.

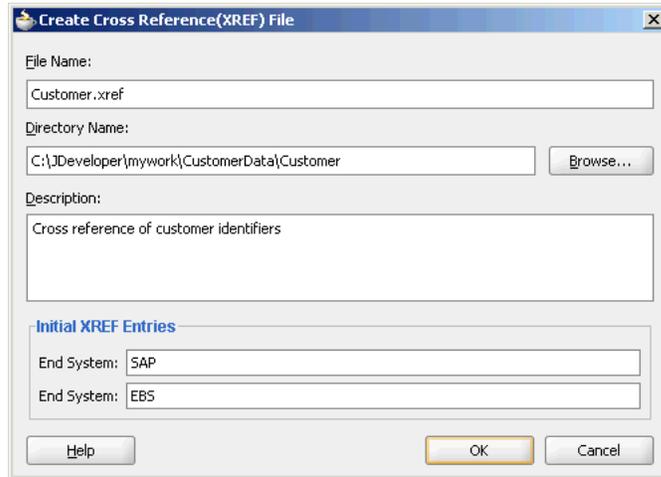
A cross reference name is used to identify an cross reference table uniquely. Two cross reference tables cannot have same name in the cross reference repository. The cross reference filename is the name of the cross reference table with an extension of *.xref*.

7. In the **Description** field, enter a description for the cross reference. For example, *Cross reference of Customer identifiers*.
8. Specify end system names in the End System fields.

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.

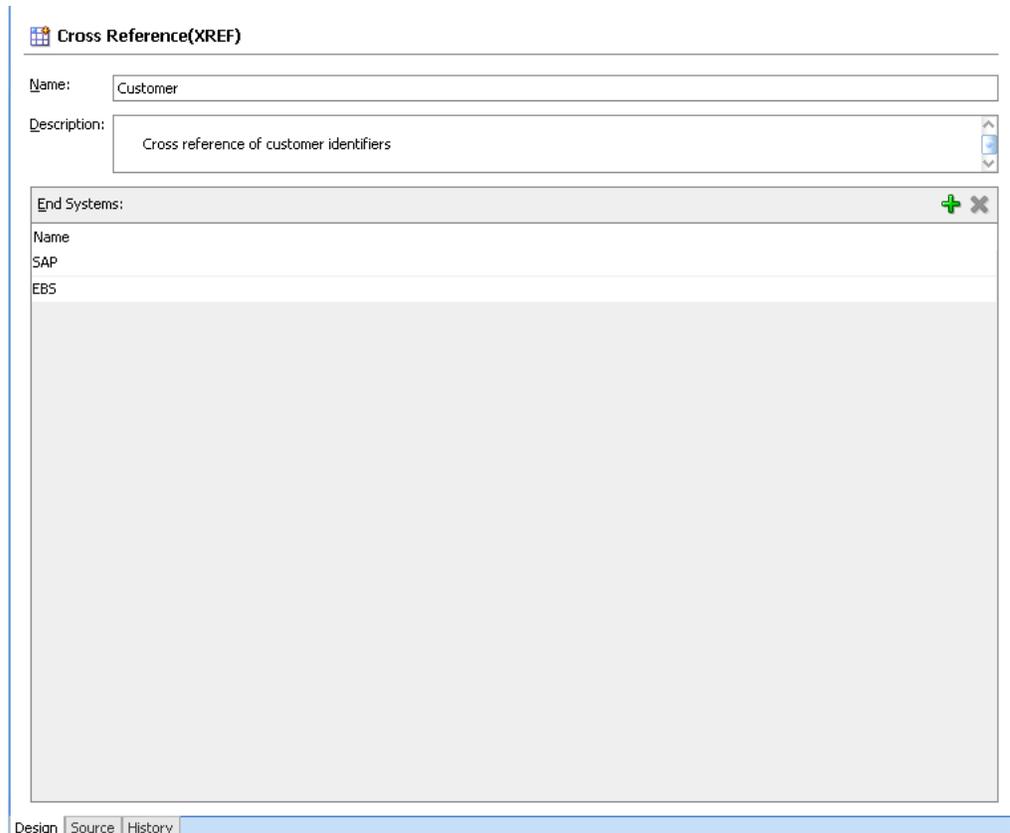
A sample Create Cross Reference(XREF) File dialog is displayed in [Figure 10–4](#).

Figure 10–4 Create Cross Reference(XREF) File Dialog



9. Click **OK**. The Cross Reference(XREF) editor is displayed, as shown in [Figure 10–5](#). You can use this editor to modify the cross reference.

Figure 10–5 Cross Reference Editor



Note: -Cross reference table names, cross reference column names, and cross reference values are case-sensitive.

10.2.2 Adding a Column to a Cross Reference Table

Perform the following steps to add a column to a cross reference table:

1. Click **Add**. A new column is added.
2. Double-click the newly added column.
3. Enter the column name. For example, SBL.

10.2.3 Deleting a Column from a Cross Reference Table

Perform the following steps to delete a column from a cross reference table:

1. Select the Column that you want to delete.
2. Click **Delete**. The selected column is deleted from the cross reference table.

10.3 Populating Cross Reference Tables

A cross reference table needs be populated at run time before using it. This can be done by using the following XPath extension functions:

- [xref:populateXRefRow Function](#)
- [xref:populateXRefRow1M Function](#)

10.3.1 xref:populateXRefRow Function

You can use the `xref:populateXRefRow` function to populate a cross reference column with a value. This function returns a string value which is the cross reference value being populated. 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 10-3](#) describes these modes.

Table 10–3 *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", "SAP", "SAP_100", "Common", "CM001", "ADD")</code> adds the reference value SAP_100 in the SAP 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.
LINK	<p>Adds the cross reference value corresponding to the existing reference value. For example, <code>xref:populateXRefRow("customers.xref", "SAP", "SAP_100", "Common", "CM001", "Link")</code> links the value CM001 in the Common column to the SAP_100 value in the SAP 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 reference value is not found. ■ The value being linked exists in that column for that table.
UPDATE	<p>Updates the cross reference value corresponding to an existing reference column-value pair. For example, <code>xref:populateXRefRow("customers.xref", "SAP", "SAP_100", "SAP", "SAP_1001", "Update")</code> updates the value SAP_100 in the SAP column to value SAP_1001.</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. ■ 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 10-3](#).

[Table 10-4](#) describes the `xref:populateXRefRow` function modes and exception conditions for these modes.

Table 10-4 *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
UPDATE	Absent	Absent	Exception
	Present	Absent	Exception
	Present	Present	Success

10.3.1.1 Using `xref:populateXRefRow` Function

The `xref:populateXRefRow` can be used in transformation to populate a column of a cross reference table by performing the following steps:

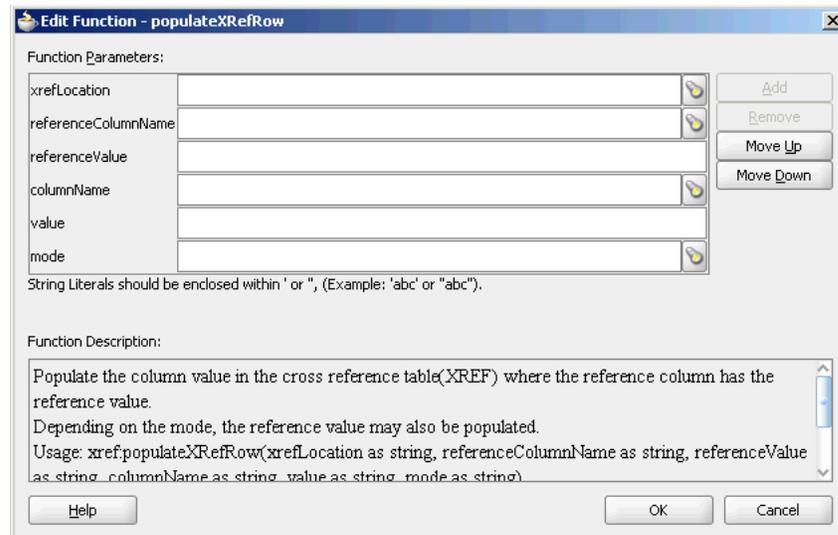
1. In the XSLT Mapper dialog, expand the trees in the Source and Target panes.
2. Drag and drop the source element to the target element.
3. In the Component Palette, click down arrow and then select **Advanced**.
4. Select **XREF Functions**.
5. Drag and drop **populateXRefRow** onto the line that connects the source object to the target object.

A `populateXRefRow` icon appears on the connecting line.

6. Double-click the **populateXRefRow** icon.

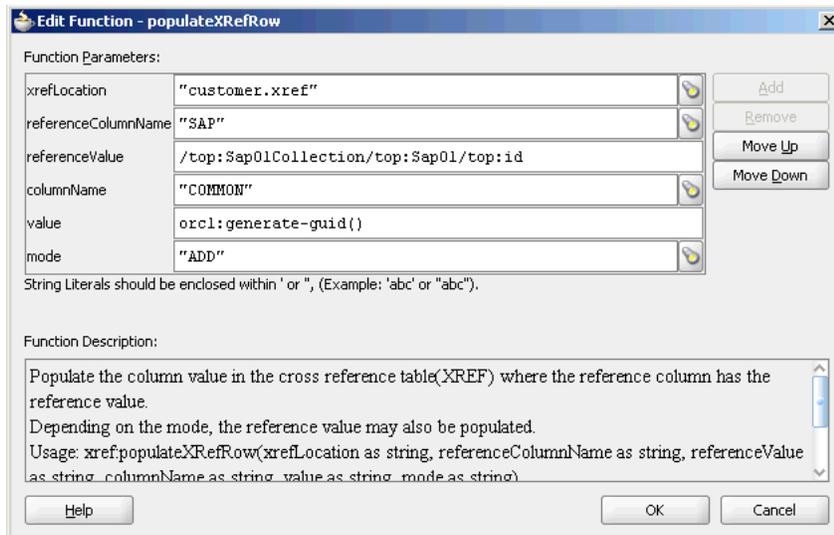
The Edit Function – `populateXRefRow` dialog is displayed, as shown in [Figure 10-6](#).

Figure 10–6 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** 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 10–7](#).

Figure 10–7 Populated Edit Function – populateXRefRow Dialog



10.3.2 xref:populateXRefRow1M Function

Many a times two values in a system can correspond to a single value in another system. For example, as shown in [Table 10–5](#), SAP_001 and SAP_0011 values refer to one value of the EBS and the SBL application.

Table 10–5 Cross Reference Table with Multiple Column Values

SAP	EBS	SBL
SAP_001	EBS_1001	SBL001
SAP_0011		
SAP_002	EBS_1002	SBL002

To populate a column in the cross reference table with multiple values, you can use the `xref:populateXRefRow1M` function. 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 10–6](#) describes these modes:

Table 10–6 *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", "SAP", "SAP_100", "Common", "CM002", "ADD")</code> adds the reference value SAP_100 in the reference column SAP and the value CM002 in the Common 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", "SAP", "SAP_100", "Common", "CM001", "Link")</code> links the value CM001 in the Common column to the SAP_100 value in the SAP 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 10–7 describes the `xref:populateXRefRow1M` function modes and exception conditions for these modes.

Table 10–7 *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

The design time steps for using the `xref:populateXRefRow1M` function are similar to the `xref:populateXRefRow` function described in ["Using xref:populateXRefRow Function"](#).

10.4 Looking Up Cross Reference Tables

After populating the cross reference table, you can use it to lookup for a value. This can be done by using the following XPath extension functions:

- [xref:lookupXRef Function](#)
- [xref:lookupXRef1M Function](#)

10.4.1 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 10-2](#) 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:lookupXRefRow` 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

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.

10.4.1.1 Using xref:lookupXRef Function

You can use the `xref:lookupXRef` function to look up a cross reference table column by performing the following steps during transformation:

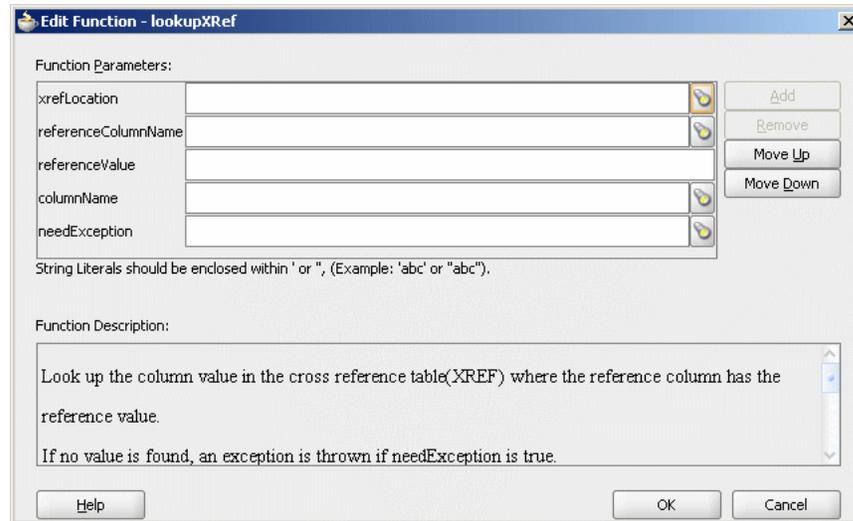
1. In the XSLT Mapper dialog, expand the trees in the Source and Target panes.
2. Drag and drop the source element to the target element.
3. In the Component Palette, click the down arrow and then select **Advanced**.
4. Select **XREF Functions**.
5. Drag and drop **lookupXRef** onto the line that connects the source object to the target object.

A `lookupXRef` icon appears on the connecting line.

6. Double-click the **lookupXRef** icon.

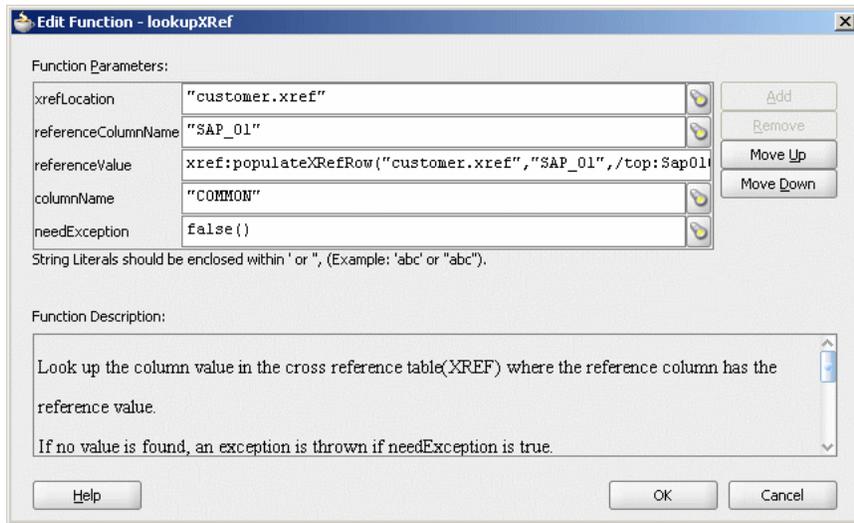
The Edit Function – lookupXRef dialog is displayed, as shown in [Figure 10–8](#).

Figure 10–8 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 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 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 10–9](#).

Figure 10–9 Populated Edit Function – lookupXRef Dialog



10.4.2 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. This 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 10–5](#) for multiple values corresponding to EBS_1001 value in the EBS column:

```
xref:lookupXRef1M("customers.xref", "EBS", "EBS_1001", "Common", 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 value is set to `true`, an exception is thrown if the value is not found else an empty value 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.

The design time steps for using the `xref:lookupXRef1M` function are similar to the `xref:lookupXRef` function explained in "[Using xref:lookupXRef Function](#)" on page 10-12.

10.5 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 was successful else returns `false`.

A cross reference table row should have at least two mappings. Therefore, 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.

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.

However, using the column value marked for delete as a reference value in the `LINK` mode of `xref:populateXRefRow` function, would raise an error.

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.

Perform the following steps to delete a value from a cross reference table column:

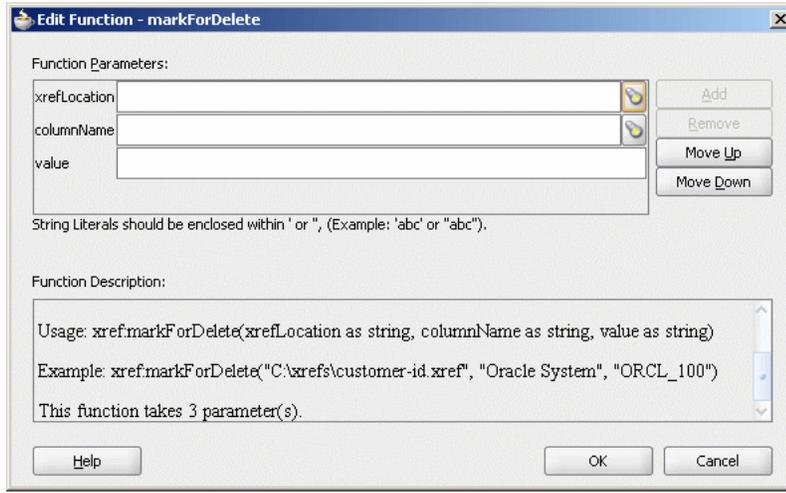
1. In the XSLT Mapper dialog, expand the trees in the Source and Target panes.
2. Drag and drop the source element to the target element.
3. In the Component Palette, click the down arrow and then select **Advanced**.
4. Select **XREF Functions**.
5. Drag and drop **markForDelete** onto the line that 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 10–10](#).

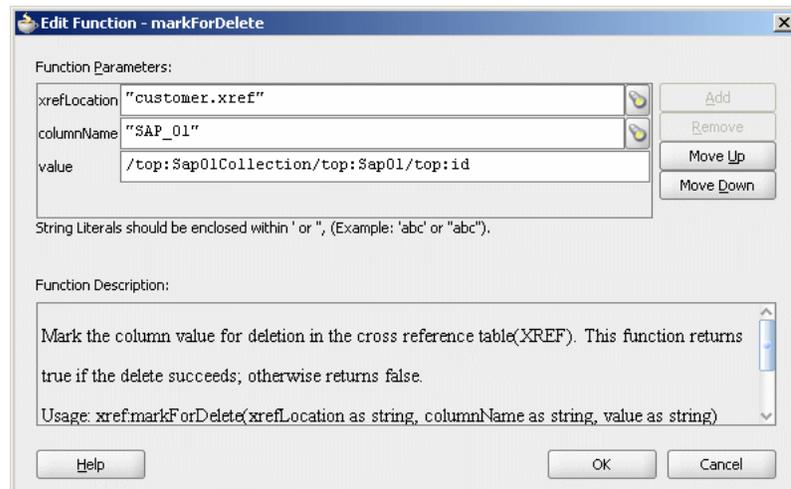
Figure 10–10 Edit Function – markForDelete Dialog



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 10–11](#).

Figure 10–11 Populated Edit Function – markForDelete Dialog



8. Click OK.

10.6 Schema Definition(XSD) File for Cross References

[Example 10–1](#) shows the cross reference XSD file. All imported cross reference XML files are validated against this schema definition file. All functions in the schema definition file should be in the following namespace:

<http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions>

Example 10–1 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>
</element>
</sequence>
</complexType>
</schema>

```

10.7 Cross Reference Use Case

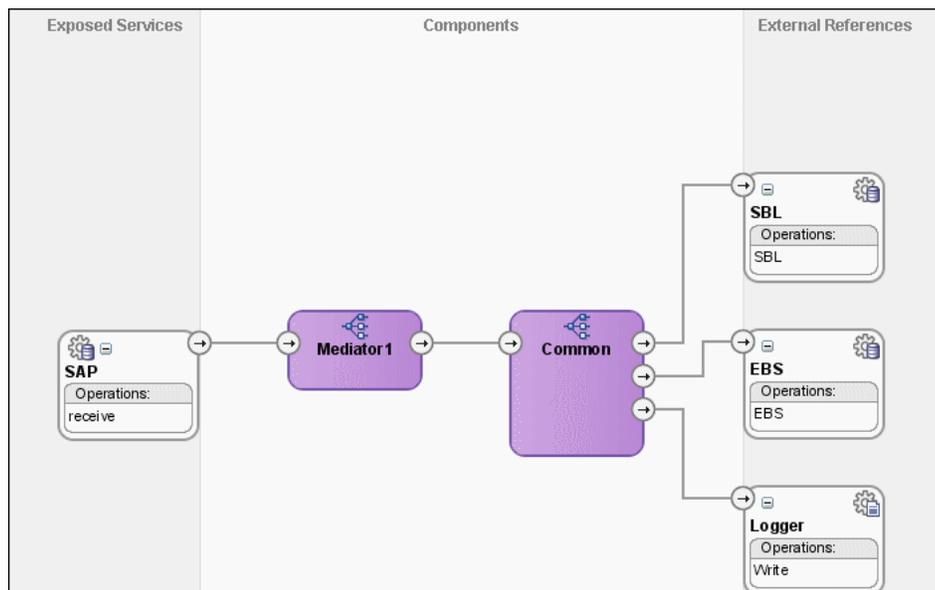
This section provides step-by-step instructions for creating and running the cross reference use case.

10.7.1 Introduction

This use case implements an integration scenario between instances of Oracle EBS, SAP and Siebel. When an insert, update, delete operation is performed on the SAP_01 table, the corresponding data is inserted or updated in the EBS and SBL tables.

[Figure 10-12](#) provides an overview of this use case.

Figure 10-12 XrefCustApp Use Case in SOA Composite Editor



The sample files for this use case are available in the `mediator-109-XRef` folder of `samples` folder.

10.7.2 Prerequisites

Following are the prerequisites for creating this use case:

- Installed and configured Oracle SOA Suite, Oracle JDeveloper, and an Oracle Database to use as your repository.
- Started Oracle SOA Suite by running the version of `startsoa` for your operating system from the `ORACLE_HOME/bin` directory.

10.7.3 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.

- [Task 1: Configuring Oracle Database and Database Adapter](#)
- [Task 2: Creating an Oracle JDeveloper Application and Project](#)
- [Task 3: Creating a Cross Reference](#)
- [Task 4: Creating a Database Adapter Service](#)
- [Task 5: Creating EBS and SBL External References](#)
- [Task 6: Creating Logger External Reference](#)
- [Task 7: Creating Mediator Components](#)
- [Task 8: Specifying Routing Rules for Mediator1](#)
- [Task 9: Specifying Routing Rules for Common Mediator](#)
- [Task 10: Configuring Oracle Application Server Connection](#)
- [Task 11: Deploying the Composite Application](#)

10.7.3.1 Task 1: Configuring Oracle Database and Database Adapter

Perform the following steps to configure Oracle Database and the Database adapter:

1. This use case uses `SCOTT` database account with password `TIGER`. You need to ensure that the `SCOTT` account is unlocked.

You can log in as `SYSDBA` and then run the `setup_user.sql` script available in the `XrefCustApp/sql` folder to unlock the account.
2. Run the `create_schema.sql` script available in the `XrefCustApp/sql` folder to create the tables required for this use case.
3. Run the `create_app_procedure.sql` script available in the `XrefCustApp/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, under the `scott` schema, to create a Cross Reference table to store runtime Cross Reference data.
5. Edit the `oc4j-ra.xml` file available in the `OH/j2ee/oc4j_soa/application-deployments/default/DbAdapter` folder as follows:

```
connector-factory location="eis/DB/DBConnection1"
```

```

connector-name="Database Adapter">
  <config-property name="xDataSourceName" value="jdbc/DBConnection1"/>
  <config-property name="dataSourceName" value="" />
  <config-property name="platformClassName"
value="oracle.toplink.platform.database.Oracle9Platform" />
  <config-property name="usesNativeSequencing" value="true" />
  <config-property name="sequencePreallocationSize" value="50" />
  <config-property name="defaultNChar" value="false" />
  <config-property name="usesBatchWriting" value="false" />
</connector-factory>

```

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.

6. Edit the `data-sources.xml` available in the `OH/j2ee/oc4j_soa/config` folder to have the corresponding JNDI name for `datasource jdbc/DBConnection1` as specified in the Database Adapter's `oc4j-ra.xml`.

```

<managed-data-source connection-pool-name="XRefDemo-ConnectionPool"
jndi-name="jdbc/DBConnection1" name="DBConnection1" />
  <connection-pool name="XRefDemo-ConnectionPool">
    <connection-factory factory-class="oracle.jdbc.pool.OracleDataSource"
user="scott" password="tiger" url="jdbc:oracle:thin:@host:port:service" />
  </connection-pool>

```

This sample uses the `scott` schema.

10.7.3.2 Task 2: Creating an Oracle JDeveloper Application and Project

Perform the following steps to create an application and a project for the use case:

1. Start Oracle JDeveloper.
2. In the upper left panel, click the **Applications Navigator** tab.
3. Right-click **Applications**, then select **New Application**.
The Create Application dialog is displayed.
4. In the **Application Name** field, enter `XrefCustApp`, and then click **OK**.
The Create Project dialog is displayed.
5. In the **Project Name** field, enter `XrefCustApp` and click **OK**.
6. Right-click in the Applications Navigator pane and select **New**.
The New Gallery dialog is displayed.
7. From the **Categories** navigator, select **SOA Tier**.
8. From the **Items** list, select **SOA Composite** and click **OK**.
The Create SOA Composite dialog is displayed.
9. From the Composite Template list, select **Empty Composite** and then click **OK**.
The Applications Navigator of Oracle JDeveloper is updated with the new application and project and the Design tab contains, a blank palette.
10. From the **File** menu, click **Save All**.

10.7.3.3 Task 3: Creating a Cross Reference

Perform the following steps to create a cross reference:

1. Right-click the project and select **New**.

2. Select **SOA Tier** from Categories and then select **Transformations**.
3. Select **Cross Reference(XREF)** from Items.
4. Click **OK**.

The Create Cross Reference(XREF) File dialog is displayed.

5. Enter `customer` in the **File Name** field.
6. Enter `SAP_01` in one **End System** field and `EBS_i76` in another.
7. Click **OK**.

The Cross Reference Editor is displayed.

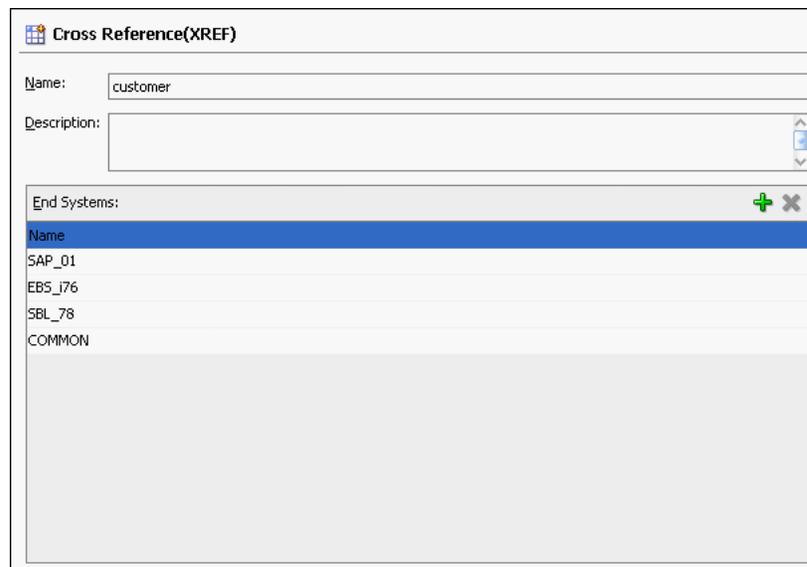
8. Click **Add**.

A new row is added.

9. Enter `SBL_78` as End System name in the newly added row.
10. Click **Add** and enter `Common` as End System name.

The Cross Reference Editor would appear as shown in [Figure 10–13](#).

Figure 10–13 Customer Cross Reference



11. From the **File** menu, click **Save All** and close the Cross Reference Editor.

10.7.3.4 Task 4: Creating a Database Adapter Service

Perform the following steps to create a Database adapter service, named `SAP`:

1. From the Components 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`.

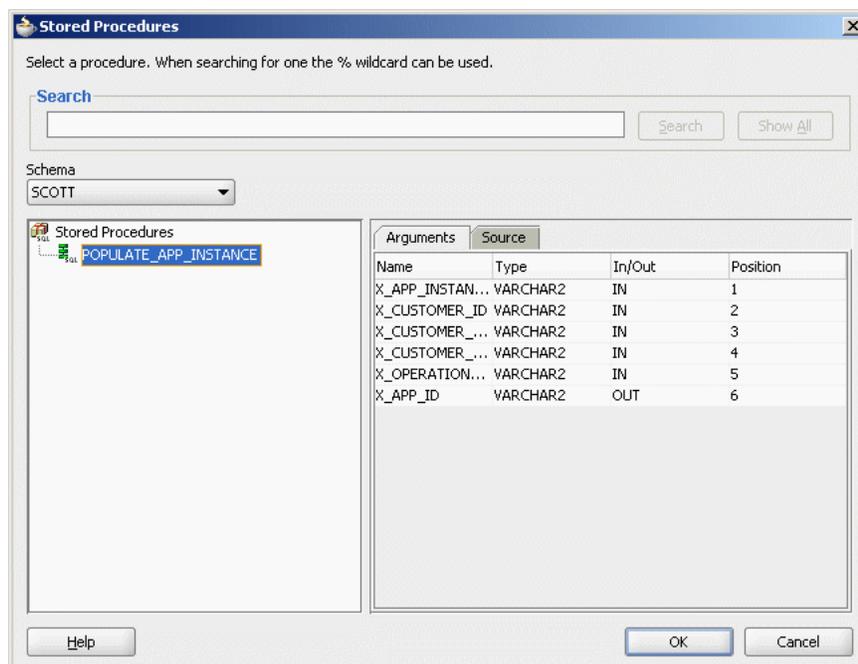
[Figure 10-14](#) shows the Logical Delete page of the Adapter Configuration Wizard.

10.7.3.5 Task 5: Creating EBS and SBL External References

Perform the following steps to create two external references named EBS and SBL:

1. From the Components 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 10–16](#).

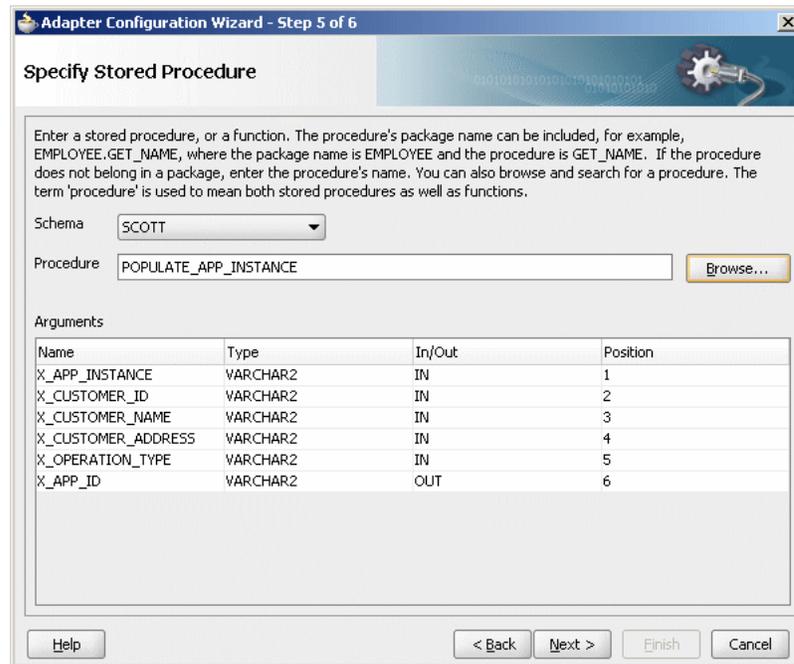
Figure 10–16 Stored Procedure Dialog



13. Click **OK**.

The Specify Stored Procedure page appears as shown in [Figure 10–17](#).

Figure 10–17 Specify Stored Procedure Page of Adapter Configuration Wizard



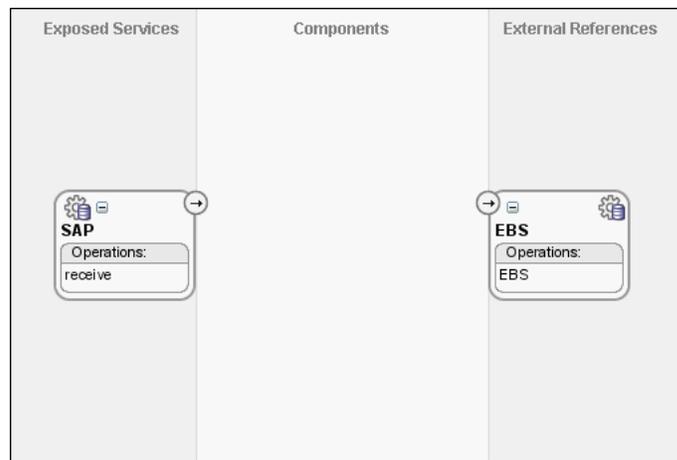
14. Click **Next**.

The Finish page is displayed.

15. Click **Finish**.

[Figure 10–18](#) shows the EBS reference in SOA Composite Editor.

Figure 10–18 EBS Reference in SOA Composite Editor

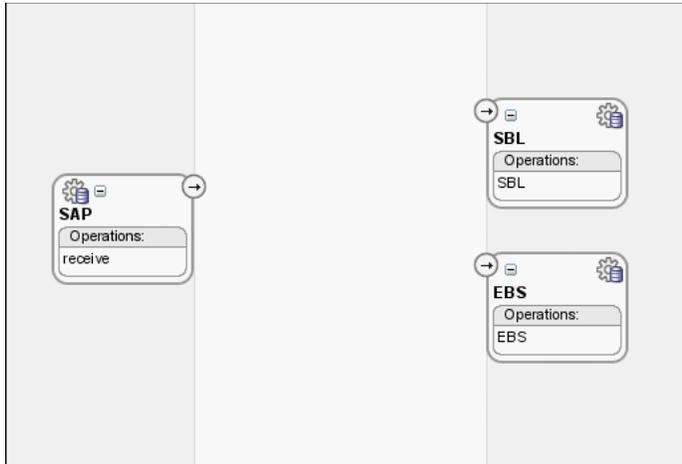


16. From the **File** menu, click **Save All**.

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 10–19](#).

Figure 10–19 SBL Reference in SOA Composite Editor



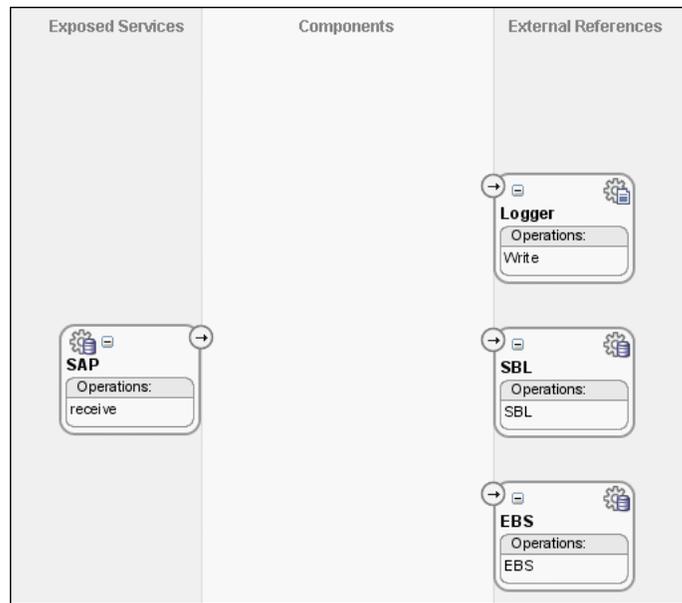
10.7.3.6 Task 6: Creating Logger External Reference

Perform the following steps to create a file adapter reference, named `Logger`:

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 `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 10–20](#) shows the `Logger` reference in the SOA Composite Editor.

Figure 10–20 *Logger Reference in SOA Composite Editor*



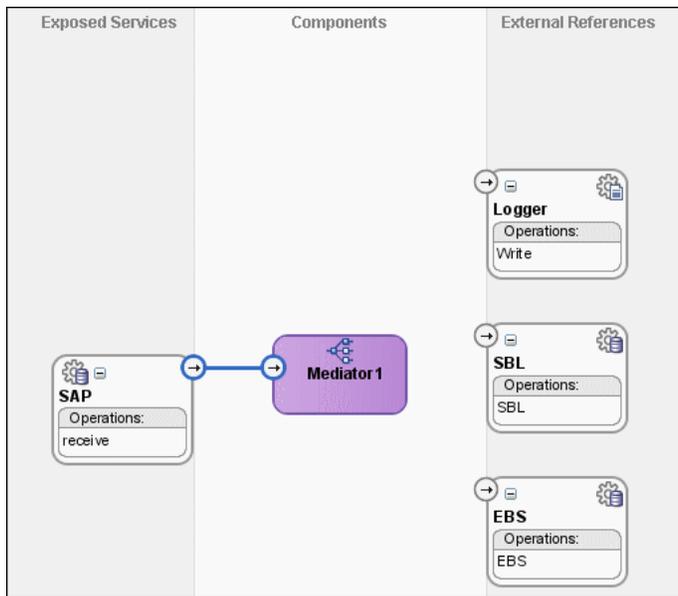
15. From the **File** menu, click **Save All**.

10.7.3.7 Task 7: Creating Mediator Components

Perform the following steps to create a Mediator named `Mediator1`:

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 10–21](#).

Figure 10–21 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.

10.7.3.8 Task 8: Specifying Routing Rules for Mediator1

You need to specify routing rules for following operations:

- [Insert](#)
- [Update](#)
- [UpdateID](#)
- [Delete](#)

Insert

Perform the following tasks 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 **Using Transformation** 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 **top:SAP01** source element to the **inp1:Customer** target element.

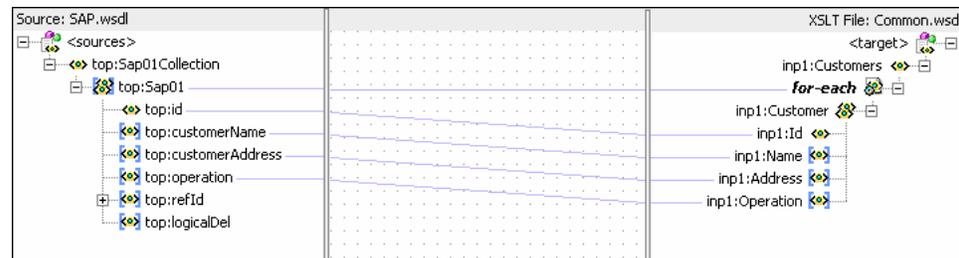
The Auto Map Preferences dialog is displayed.

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 10–22](#).

Figure 10–22 *SAP_TO_COMMON_INSERT.xml Transformation*



15. From the Components Palette, select **Advanced**.

16. Select **XREF Functions**.

17. Drag and drop **populateXRefRow** 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.

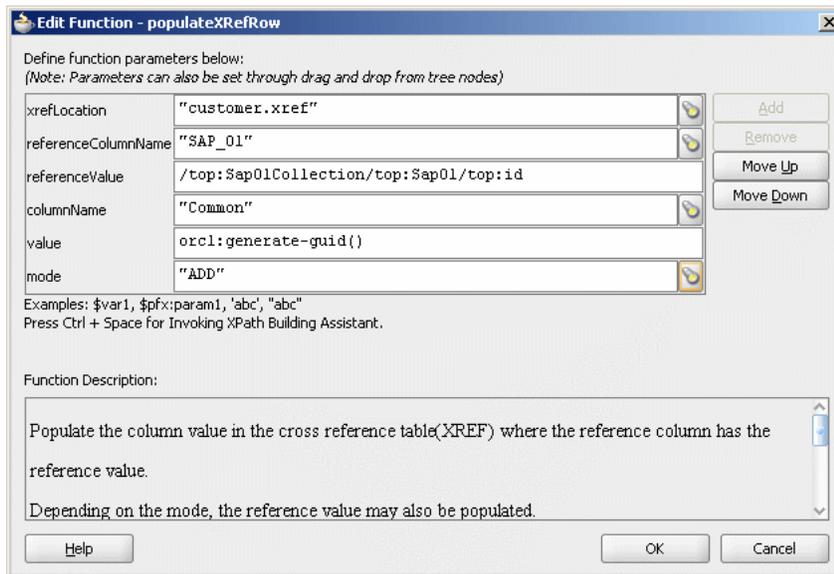
The SCA Resource Lookup dialog is displayed.

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 orcl:generate-guid().
25. In the **mode** field, enter "Add" or click **Search** to select this mode.

Figure 10-23 shows populated Edit Function – populateXRefRow dialog.

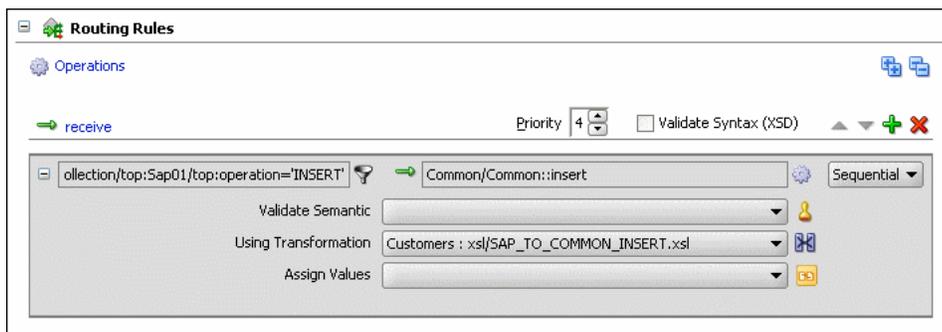
Figure 10-23 Edit Function – populateXRefRow Dialog: XrefCustApp Use Case



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 10-24.

Figure 10-24 Routing Rules Panel with Insert Operation



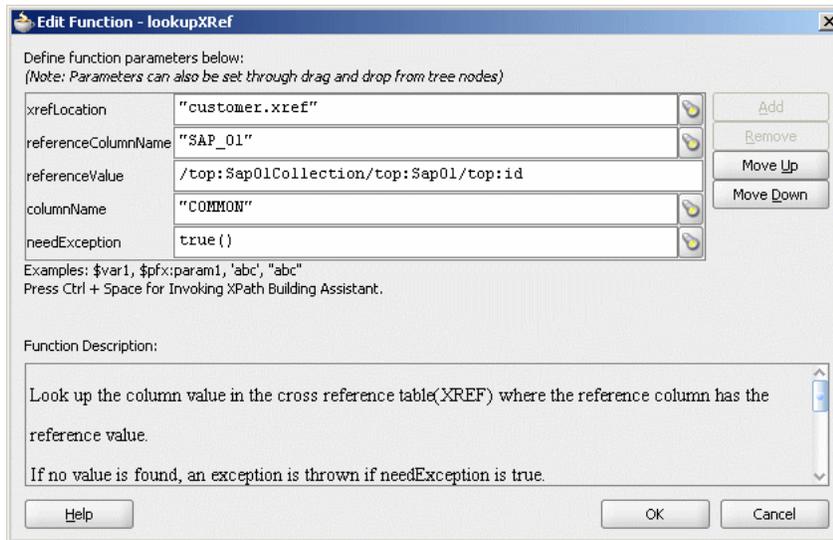
Update

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 Using Transformation 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 **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 **lookupXRef** 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 10–25](#) shows populated Edit Function – looupXRef dialog.

Figure 10–25 Edit Function – looupXRef Dialog: XrefCustApp Use Case

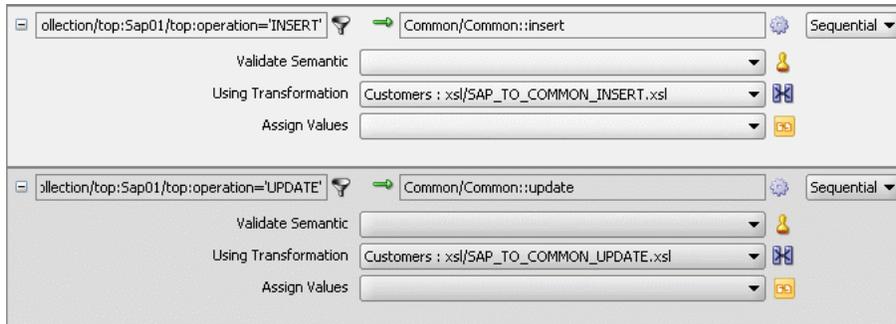


23. Click **OK**.

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 10–26](#).

Figure 10–26 Insert Operation and Update Operation



UpdateID

Perform the following tasks to create routing rules for UpdateID operation:

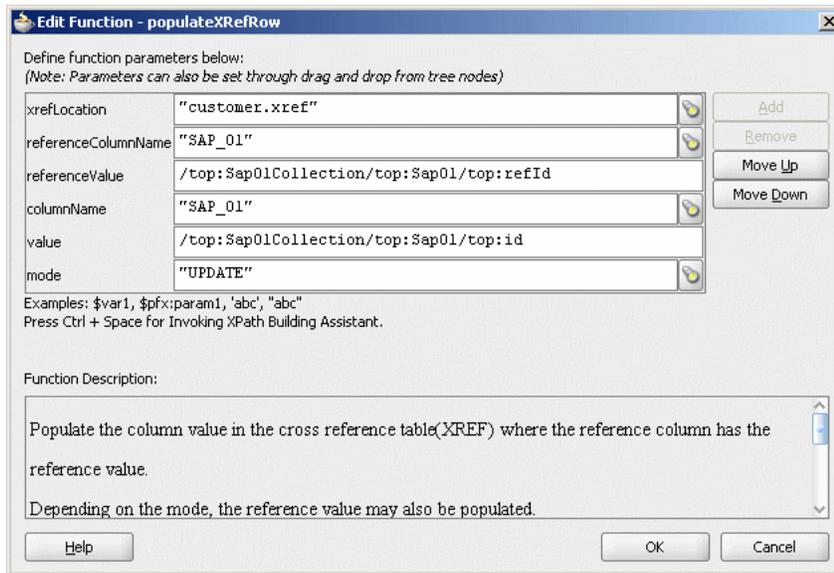
1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefCustApp, Mediators, Common, Services, Common**.
4. Select **updateid** and click **OK**.
5. Click the Filter 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 Using Transformation field.
The Request Transformation map dialog is displayed.
9. Select **Create New Mapper File** and enter `SAP_TO_COMMON_UPDATEID.xml`.
10. Click **OK**.
A `SAP_TO_COMMON_UPDATEID.xml` tab is displayed.
11. Drag and drop **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 **populateXRefRow** 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 10-27](#) shows a populated Edit Function – populateXRefRow dialog.

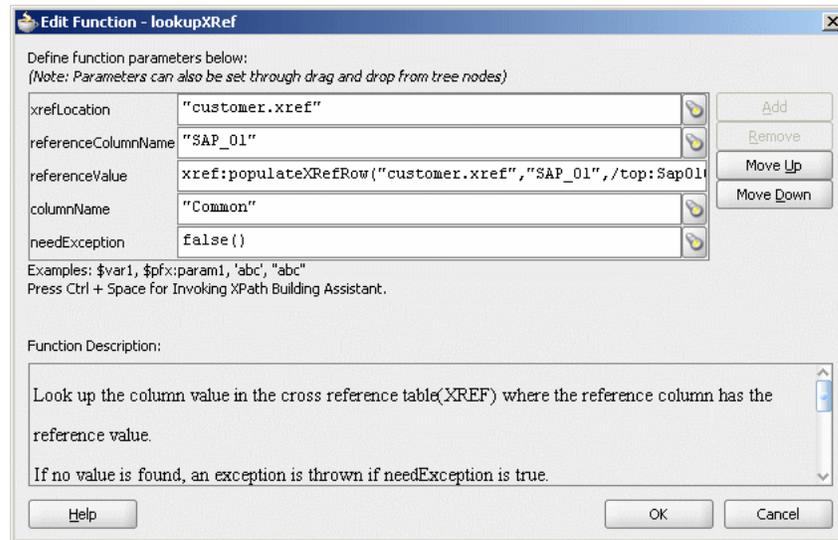
Figure 10–27 Edit Function – populateXRefRow Dialog: XrefCustApp Use Case



24. Drag and drop **lookupXRef** 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
`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 10–28](#) shows a populated Edit Function – lookupXRef dialog.

Figure 10–28 Edit Function – lookupXRef Dialog: XrefCustApp Use Case

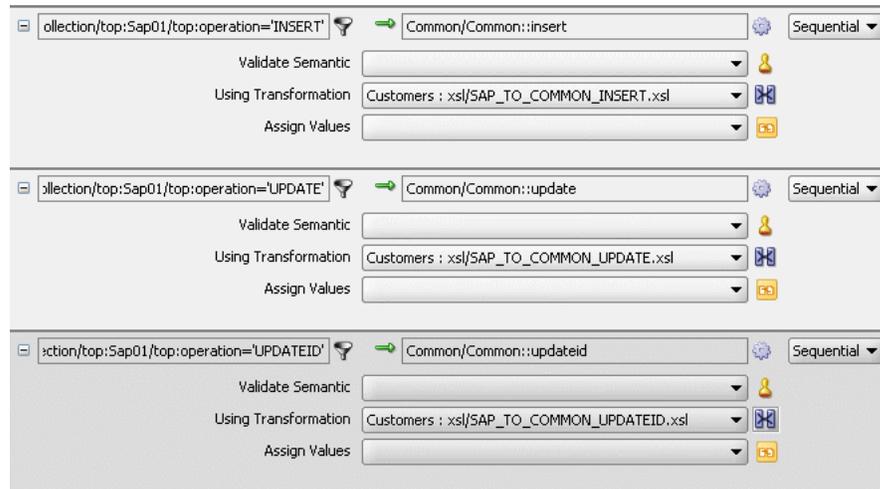


32. Click **OK**.

33. Click **Save All** and close the SAP_TO_COMMON_UPDATEID.xsl window.

The Routing Rules panel would appear as shown in [Figure 10–29](#).

Figure 10–29 Insert, Update, and UpdateID Operations



Delete

Perform the following tasks to create routing rules for Delete operation:

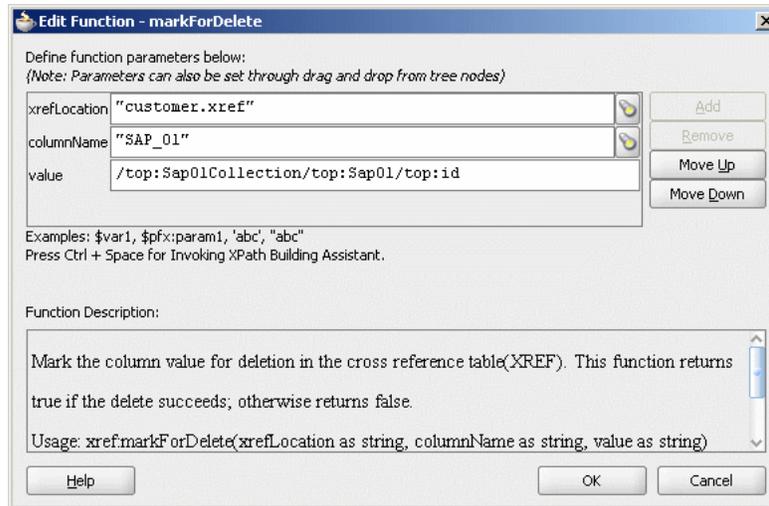
1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefCustApp, Mediators, Common, Services, Common**.
4. Select **delete** and click **OK**.

-
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 Using Transformation field.
The Request Transformation map dialog is displayed.
 9. Select **Create New Mapper File** and enter `SAP_TO_COMMON_DELETE.xsl`.
 10. Click **OK**.
A `SAP_TO_COMMON_DELETE.xsl` 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 **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 **markForDelete** from Components 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 10–30 shows a populated Edit Function – markForDelete dialog.

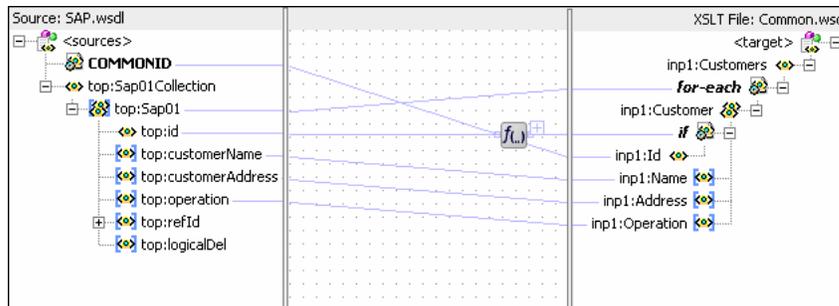
Figure 10–30 Edit Function – markForDelete Dialog: XrefCustApp Use Case



31. Click **OK**.

The `SAP_TO_COMMON_DELETE.xsl` would appear as shown in Figure 10–31.

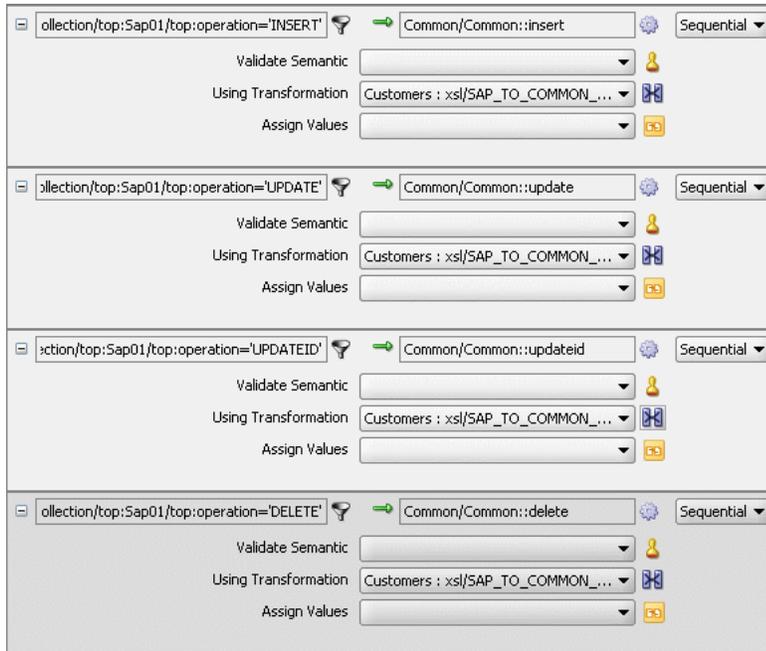
Figure 10–31 `SAP_TO_COMMON_DELETE.xsl`



32. Click **Save All** and close the `SAP_TO_COMMON_DELETE.xsl` tab.

The Routing Rules panel would appear as shown in Figure 10–32.

Figure 10–32 Insert, Update, UpdateID, and Delete Operations



10.7.3.9 Task 9: Specifying Routing Rules for Common Mediator

You need to specify routing rules for following operations of Common mediator:

- [Insert Operation](#)
- [Delete Operation](#)
- [Update Operation](#)
- [UpdateID Operation](#)

Insert Operation

Perform the following tasks 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 Using Transformation 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.

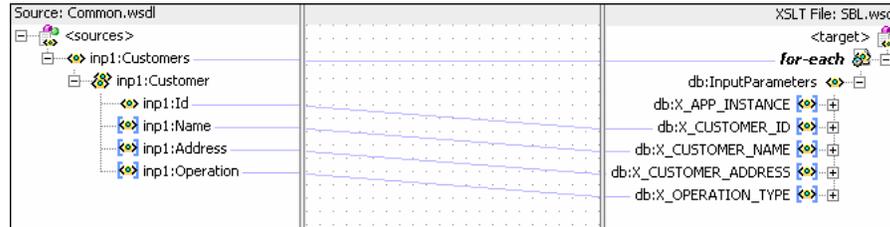
9. Drag and drop **inp1:Customers** source element to the **db:InputParameters** target element.

The Auto Map Preferences dialog is displayed.

10. Click **OK**.

The transformation is created as shown in [Figure 10–33](#).

Figure 10–33 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 Using Transformation 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 an **populateXRefRow** function from Components Palette to the connecting line.

22. Double-click the **populateXRefRow** icon.

The Edit Function-populateXRefRow dialog is displayed.

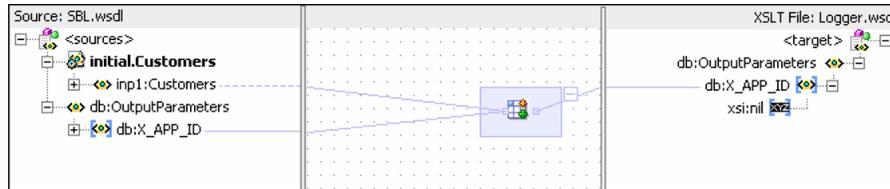
23. Enter the information in the following fields:

- **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 10-34](#).

Figure 10-34 *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')
```

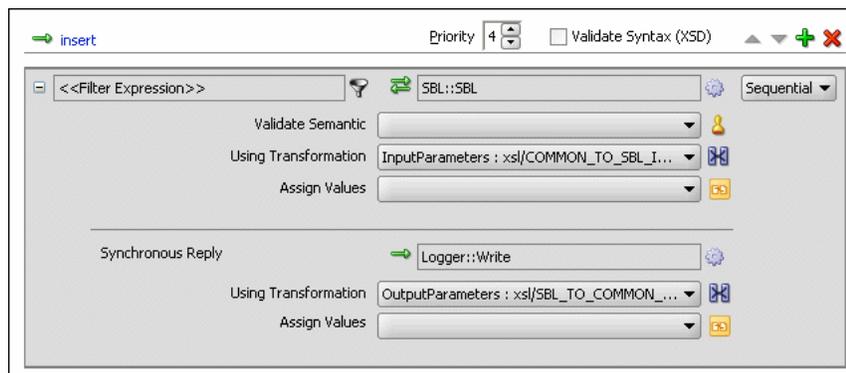
31. In the **To** section, select **Property**.

32. Select `jca.file.FileName` property and click **OK**.

33. Click **OK**.

The `insert` operation panel would appear as shown in [Figure 10-35](#).

Figure 10-35 *Insert Operation with SBL Target Service*

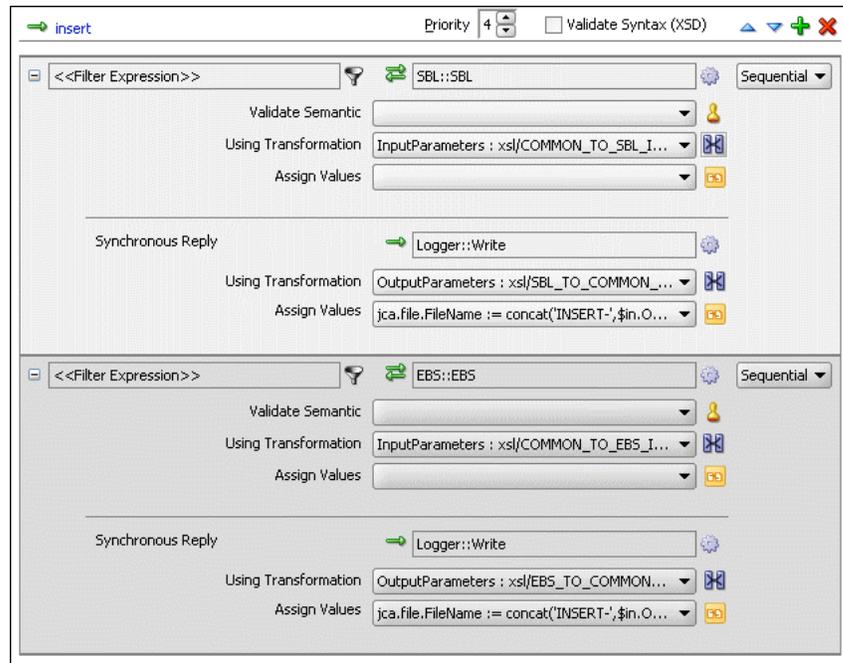


34. From the **File** menu, click **Save All**.

35. Repeat the Step 2 through Step 34 to specify another target service EBS and its routing rules.

[Figure 10-36](#) shows the `insert` operation panel with SBL and EBS target service.

Figure 10–36 Insert Operation with SBL and EBS Target Service

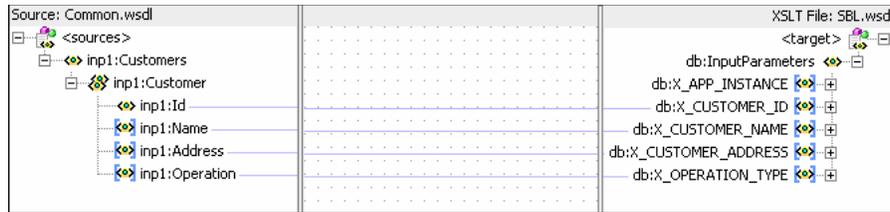


Delete Operation

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 Using Transformation 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 `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 10–33](#).

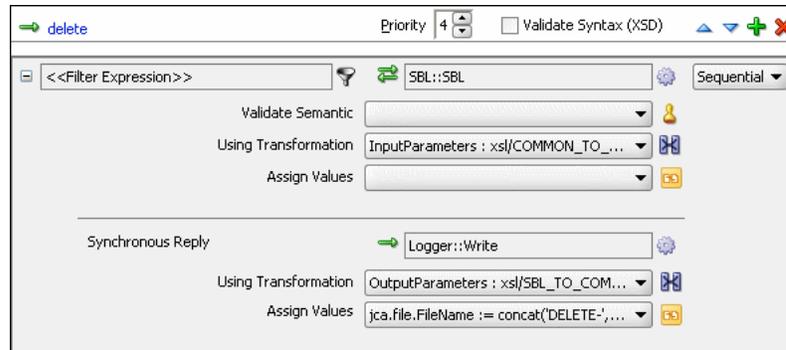
Figure 10–37 COMMON_TO_SBL_DELETE.xsl Transformation



10. Drag and drop an **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 Using Transformation 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 an **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 10–38](#).

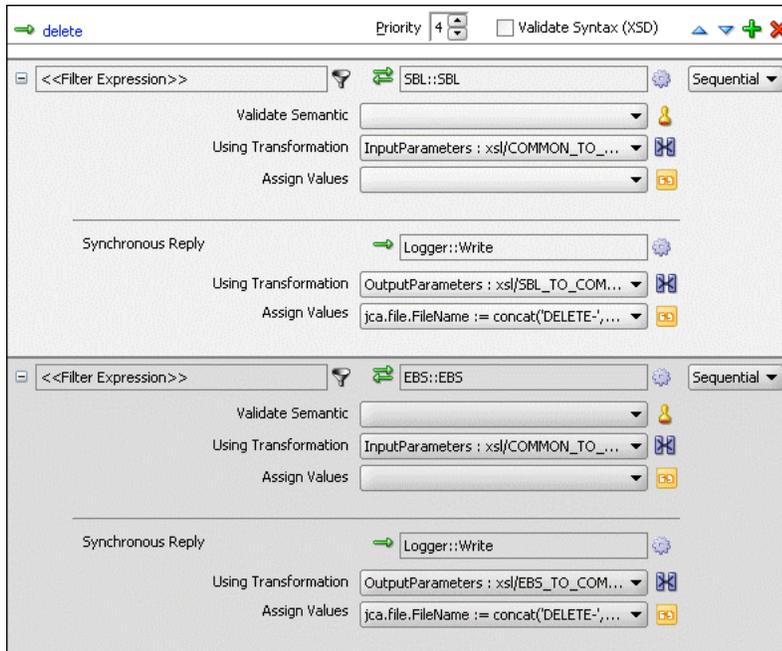
Figure 10–38 Delete Operation with SBL Target Service



- 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 10–36](#) shows the delete operation panel with SBL and EBS target service.

Figure 10–39 Delete Operation with SBL and EBS Target Service



Update Operation

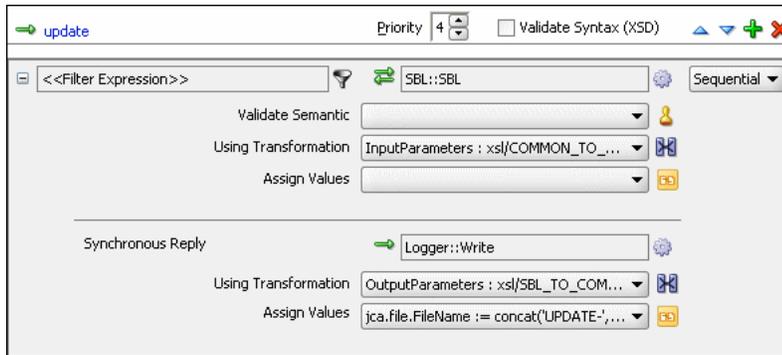
Perform the following tasks to create routing rules for Update operation:

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 Using Transformation 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 **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 10–37](#).
10. Drag and drop an **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 Using Transformation 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 10–40](#).

Figure 10–40 Update Operation with SBL Target Service

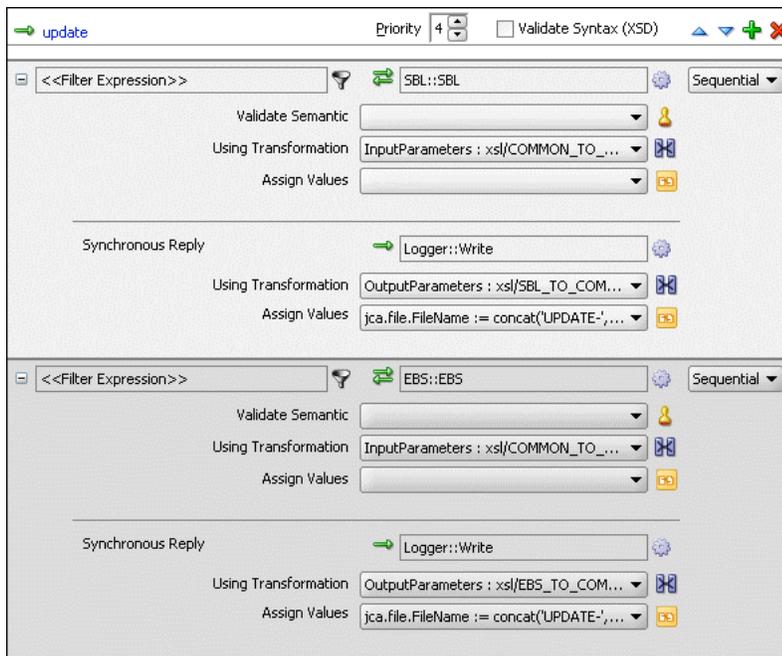


32. From the **File** menu, click **Save All**.

33. Repeat the Step 1 through Step 32 to specify another target service EBS and its routing rules.

Figure 10–41 shows the update operation panel with SBL and EBS target service.

Figure 10–41 Update Operation with SBL and EBS Target Service



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 Using Transformation 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 10-37](#).
 10. Drag and drop an **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 Using Transformation 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 an **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**.

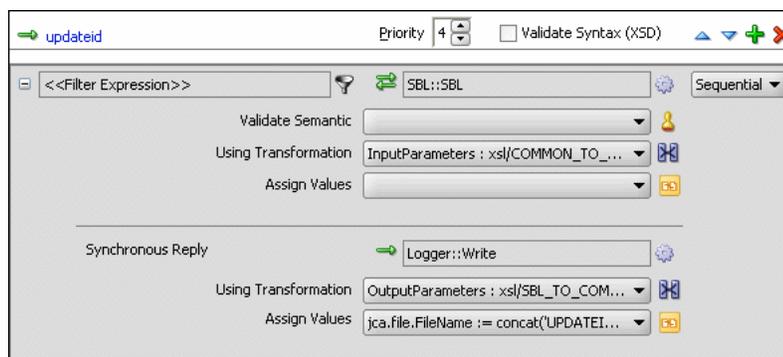
The Assign Value dialog is displayed.
30. In the **From** section, select **Expression**.
31. Click the **Invoke Expression Builder** icon.

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 10–40](#).

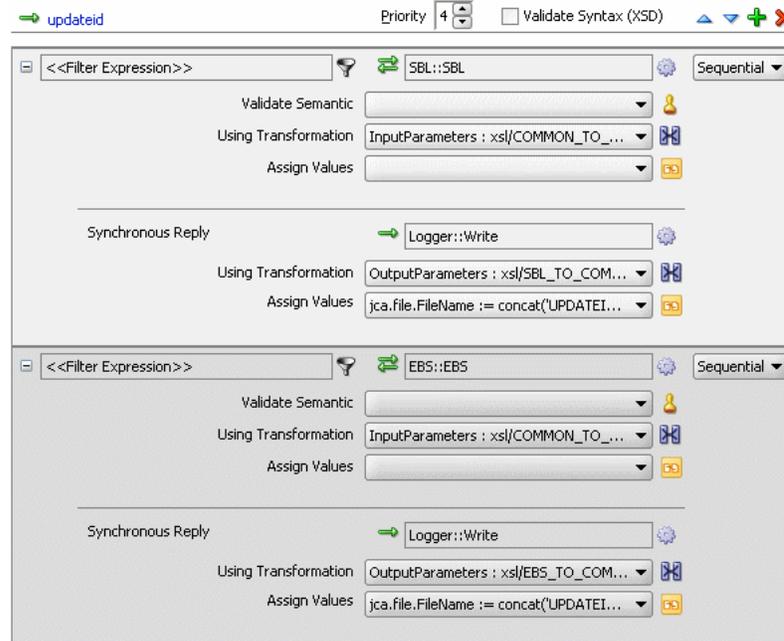
Figure 10–42 Updateid Operation with SBL Target Service



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 10–43 shows the `updateid` operation panel with SBL and EBS target service.

Figure 10–43 Updateid Operation with SBL and EBS Target Service



10.7.3.10 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 [Section 6.3.1.6, "Configuring Oracle Application Server Connection"](#).

10.7.3.11 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 Deployment Profile to Oracle Application Server

For detailed information about these steps, see [Section 3.4, "Deploying Applications"](#).

10.7.4 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.

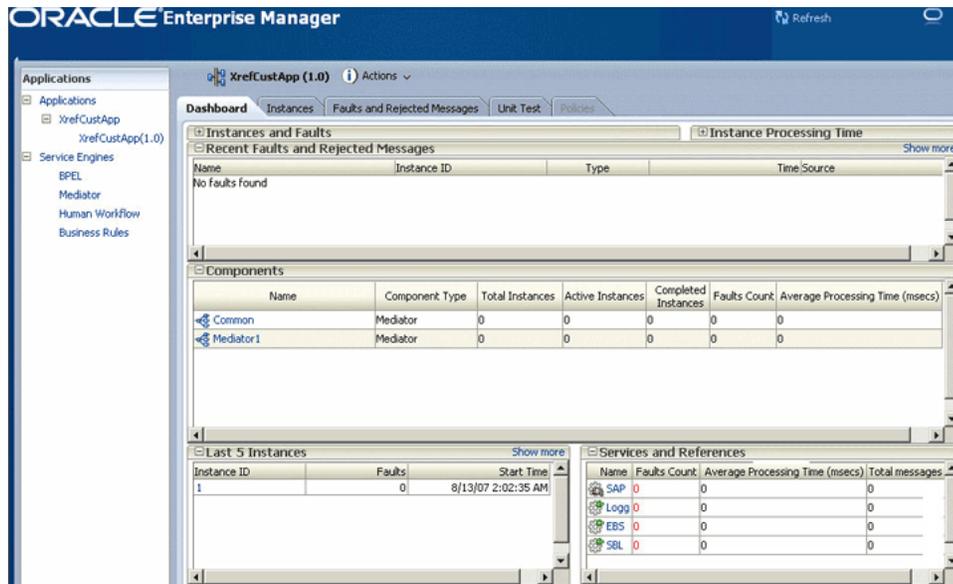
For monitoring the running instance, you can use the SOA Console at the following URL:

`http://hostname:8888/SOAConsole`

where `hostname` is the host on which you installed the Oracle SOA Suite infrastructure.

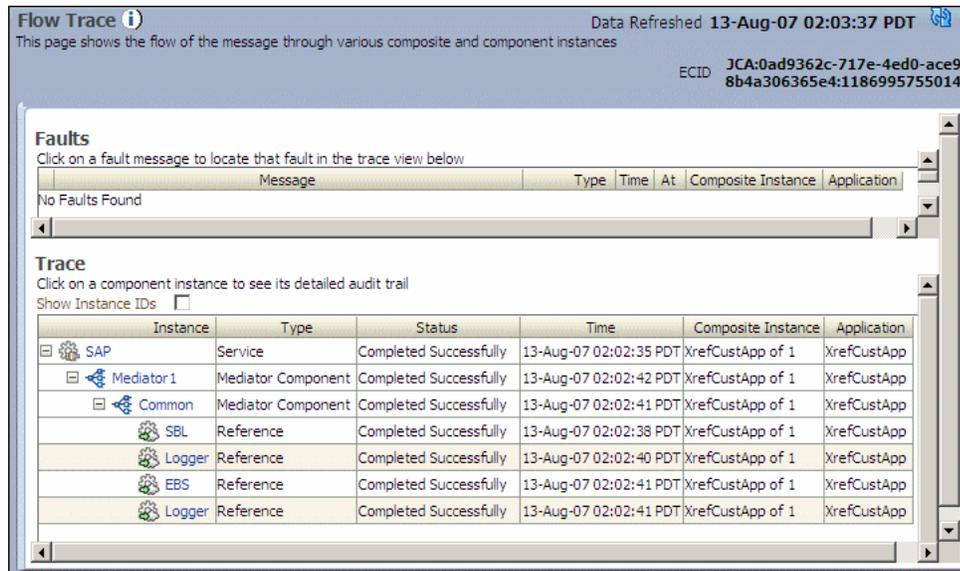
In the SOA Console, you can click the XrefCustApp to see the project dashboard as shown in Figure 10–44.

Figure 10–44 XrefCustApp Dashboard in SOA Console



To view the detailed execution trail of the XrefCustApp application, click the instance id in the instance column. The Flow Trace page is displayed as shown in Figure 10–45.

Figure 10–45 XrefCustApp Flow Trace



BPEL Process Service Component

This part describes the BPEL process service component.

This part contains the following chapters:

- [Chapter 11, "Getting Started with Oracle BPEL Process Manager"](#) through [Chapter 25, "Business Rule Service Component"](#)

Getting Started with Oracle BPEL Process Manager

This chapter describes how to start key Oracle BPEL Process Manager components, including Oracle JDeveloper and Oracle BPEL Server. An introduction to the main sections of Oracle JDeveloper that you use to design BPEL process service components is also provided. Key BPEL design features such as activities and partner links are also described.

This chapter contains the following topics:

- [Section 11.1, "Starting Oracle SOA Suite Components"](#)
- [Section 11.2, "Introduction to the BPEL Designer Environment"](#)
- [Section 11.3, "Introduction to Activities"](#)
- [Section 11.4, "Introduction to Partner Links"](#)
- [Section 11.5, "Partner Link Creation and the SOA Composite Editor"](#)
- [Section 11.6, "Introduction to Oracle BPEL Server"](#)
- [Section 11.7, "Introduction to Oracle BPEL Process Manager Technology Adapters"](#)

11.1 Starting Oracle SOA Suite Components

When you start Oracle SOA Suite, Oracle BPEL Process Manager is also started. Follow the instructions in [Table 11-1](#) to start and stop Oracle SOA Suite.

Table 11–1 Starting and Stopping Oracle BPEL Process Manager Components

To Access The...	On Windows...	On UNIX...
Oracle SOA Suite	To start Oracle SOA Server: From <code>ORACLE_HOME\bin</code> : <code>startsoa.bat</code> To stop Oracle SOA Server: From <code>ORACLE_HOME\bin</code> : <code>shutdownsoa.bat</code>	To start Oracle SOA Server: From <code>\$ORACLE_HOME/bin</code> : <code>startsoa.sh</code> To stop Oracle SOA Server: From <code>\$ORACLE_HOME/bin</code> : <code>shutdownsoa.sh</code> Note: Running <code>startsoa</code> from the operating system command prompt causes server startup messages, including errors and warnings, to display on-screen instead of being written to a log file. If you want to log these messages, redirect them to a file using standard output. For example: <code>% startsoa.bat > logfile1 &</code>
Oracle JDeveloper	Click <code>JDev_Oracle_Home\JDev\bin\jdev.exe</code> or create a shortcut	<code>\$ORACLE_HOME/jdev/bin/jdev</code>

Note: Always use the **Developer Prompt** to open an operating system command prompt when deploying services with ant. This sets all required paths. Opening an operating system command prompt in any other way is not supported.

11.2 Introduction to the BPEL Designer Environment

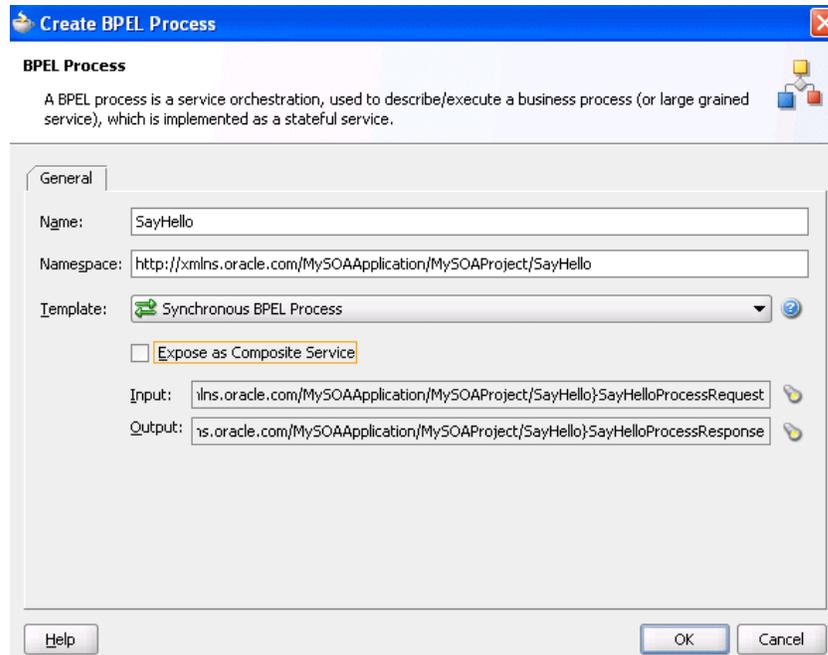
This section provides an introduction to the Oracle JDeveloper environment.

11.2.1 Introduction to BPEL Process Service Component Creation and Oracle JDeveloper

BPEL process service components are created in the SOA Composite Editor.

1. Create a BPEL process service component through one of the following methods:
 - Drag and drop a **BPEL Process** service component from the **Component Palette**.
 - Select **Composite with BPEL** in the Create SOA Composite window or Create SOA Project window.

Each method causes the Create BPEL Process window to appear.



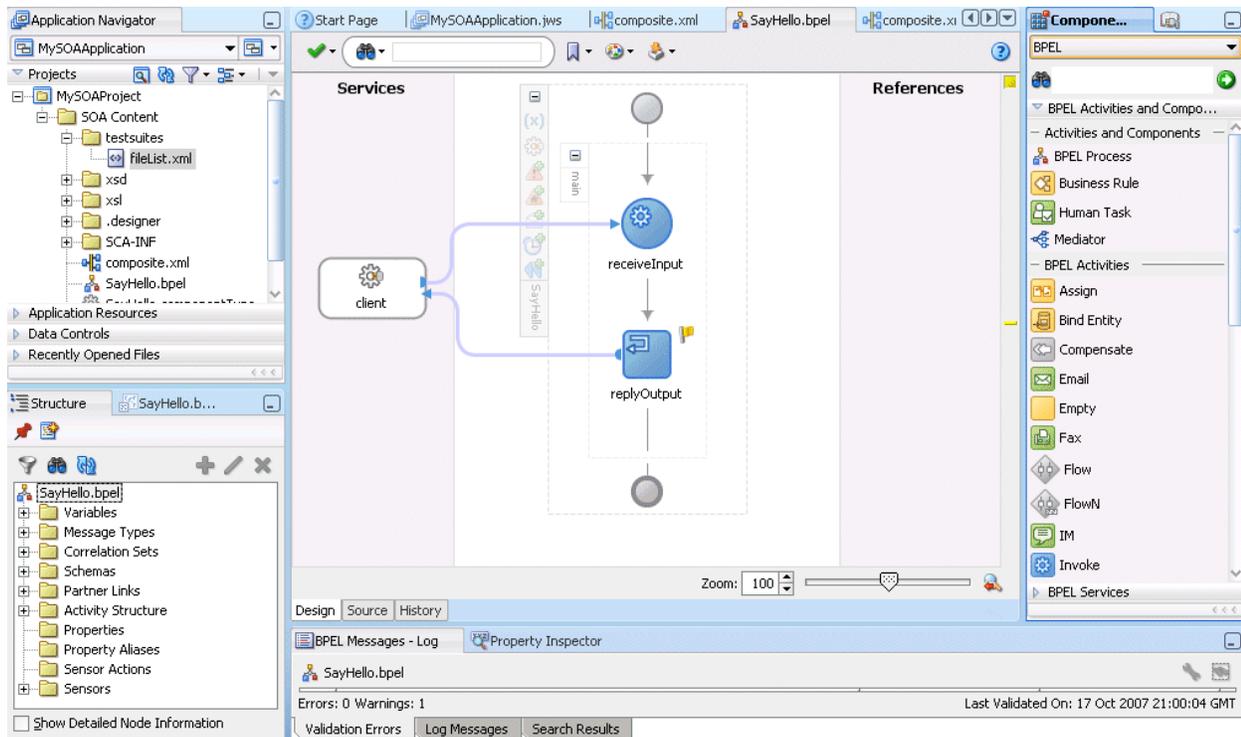
2. Provide the required details (including BPEL process name). Click **Help** for details about the types of BPEL processes you can create.

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

- A process name that begins with a number
- A process name that includes a dash (for example, **Loan-Flow**)
- Two processes with the same name, but with different capitalization

3. Click **OK**.
4. Double-click the BPEL process.

Oracle JDeveloper displays the sections shown in [Figure 11-1](#).

Figure 11–1 Oracle JDeveloper Sections

Each section of this view enables you to perform specific design and deployment tasks. [Table 11–2](#) identifies the sections listed in [Figure 11–1](#) and provides references to sections that describe their capabilities.

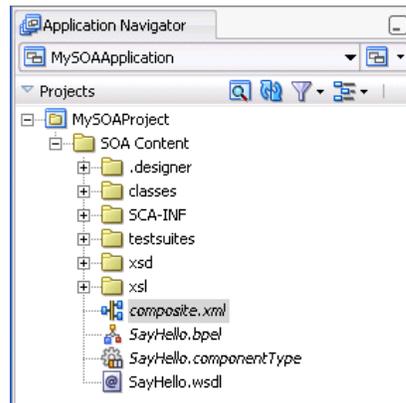
Table 11–2 Oracle JDeveloper Sections

Section	Location in Figure 11–1	See Section
Application Navigator	Upper left	Section 11.2.1.1, "Application Navigator" on page 11-5
Design window, Source window, and History window	Middle	Section 11.2.1.2, "Design Window" on page 11-5, Section 11.2.1.3, "Source Window" on page 11-7, and Section 11.2.1.4, "History Window" on page 11-8
BPEL Activities selection of the Component Palette	Upper right	Section 11.2.1.5, "Component Palette" on page 11-9
Property Inspector section	Lower right or bottom (Not shown in Figure 11–1 , but viewable by selecting Property Inspector from the View main menu.)	Section 11.2.1.6, "Property Inspector" on page 11-11
Structure Window	Lower left	Section 11.2.1.7, "Structure Window" on page 11-11
Log Window	Bottom	Section 11.2.1.8, "Log Window" on page 11-12

11.2.1.1 Application Navigator

The **Application Navigator** shown in the upper left part of [Figure 11–1](#) displays the process files. [Figure 11–2](#) 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**).

Figure 11–2 Application Navigator



[Table 11–3](#) describes these initial process files.

Table 11–3 Initial Process Files

File	Description
composite.xml	The file that describes the entire SOA composite application. See Also: Chapter 2, "Introduction to the SOA Composite Editor" for specific details about this file
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 and drop 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 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.

11.2.1.2 Design Window

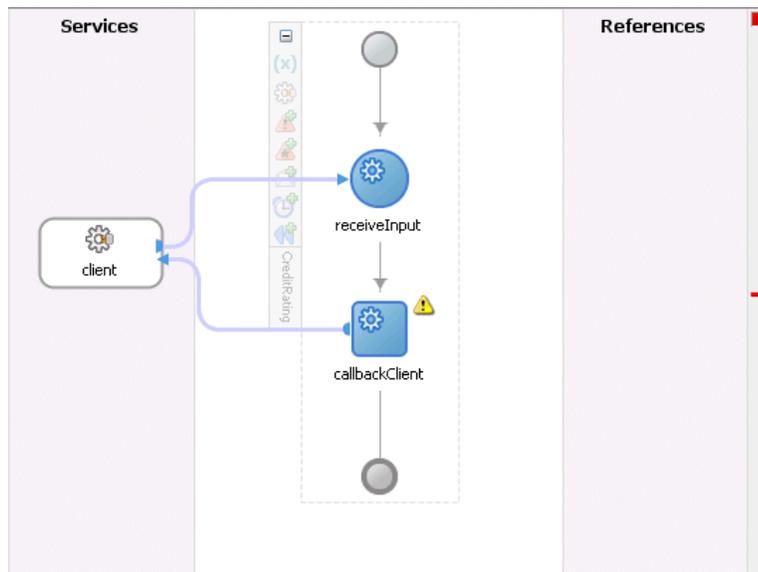
The **Design** window shown in the middle of [Figure 11–1](#) provides a visual view of the BPEL process service component that you design. This view displays when you perform one of the following actions:

- Double-click the **.bpel** file name in the **Application Navigator**

- Click the **Design** tab at the bottom of the window with the **.bpel** file selected
- Double-click the BPEL process component in the SOA composite application

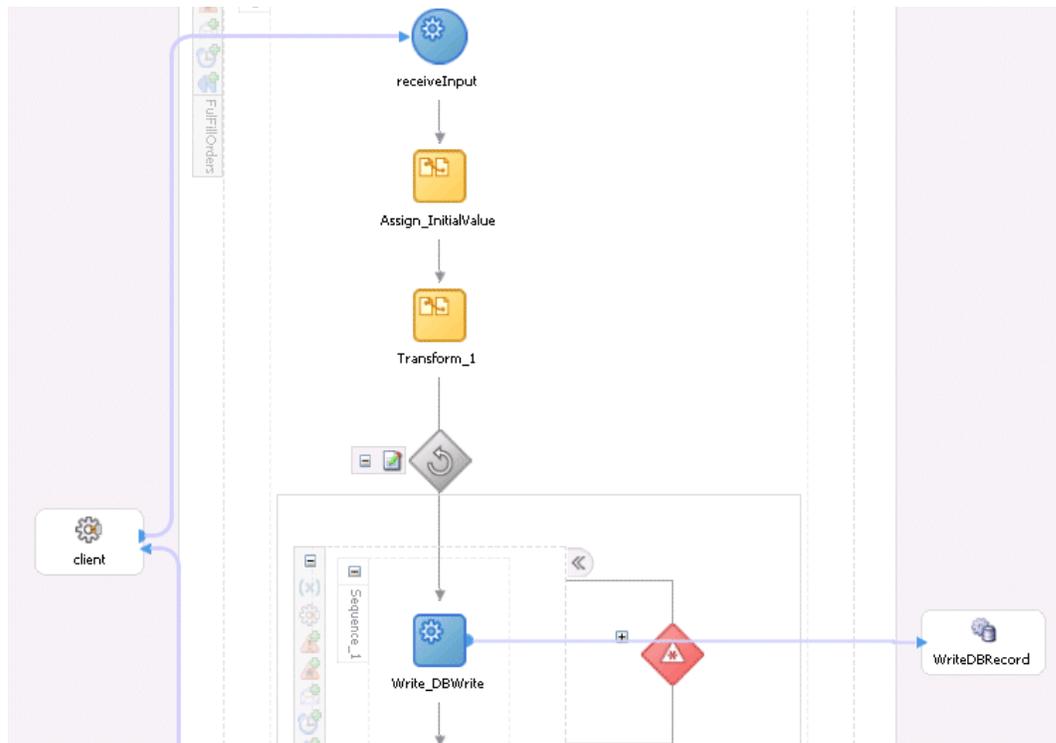
Figure 11–3 shows the activities automatically created with an asynchronous BPEL process service component. You add to the BPEL process service component by dragging and dropping activities, creating variables, creating partner links, and so on.

Figure 11–3 Design (After Creation of an Asynchronous BPEL Process Service Component)



As you design the BPEL process service component by dragging and dropping activities, creating partner links, and so on, the **Design** window changes. Figure 11–4 shows the **Design** window later in the design phase after adding a partner link (in this example, named **WriteDBRecord**) and the additional activities (**invoke**, **receive**, **assign**, **transform**, and others).

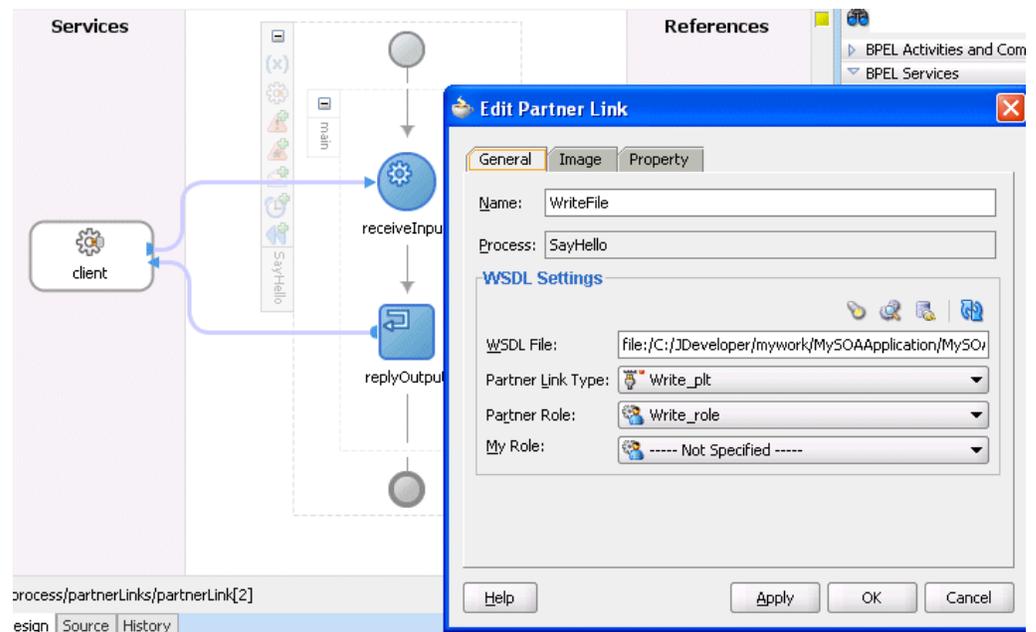
Figure 11–4 Design (After Design Phase)



11.2.1.3 Source Window

Click **Source** at the bottom to view the syntax inside the BPEL process service component files. As you drag and drop activities and partner links, and perform other tasks, the syntax in these source files is immediately updated to reflect these changes. For example, Figure 11–5 shows the property sheet as it is being edited.

Figure 11–5 CreditRatingService Partner Link Icon and Property Sheet



Click **Source** at the bottom of the window. [Figure 11–6](#) shows part of the **Source** of a **.bpel** file. Details about the **CreditRatingService** partner link you created appear in the file.

Figure 11–6 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>

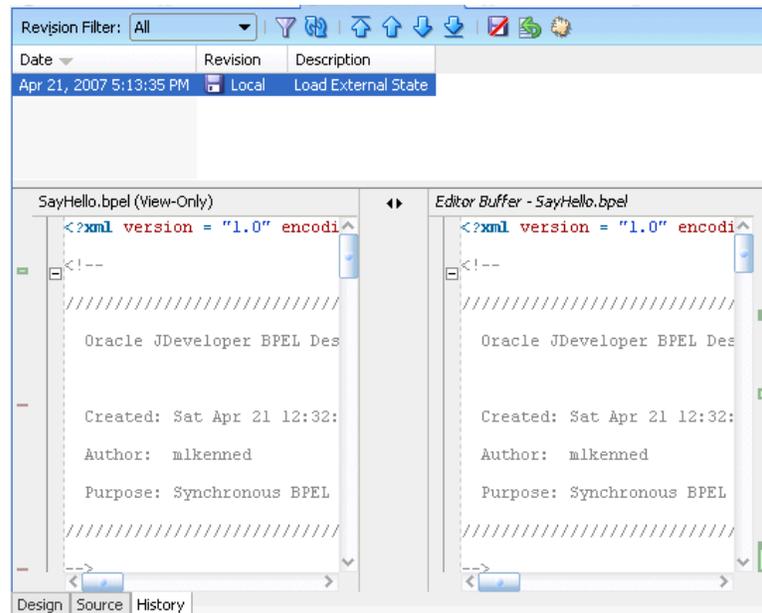
```

See Also: The following documentation for examples and descriptions of the types of syntax that appear in BPEL process service component files:

- [Chapter 12, "Manipulating XML Data in BPEL Processes"](#) through [Chapter 22, "Interaction Patterns in BPEL Processes"](#)

11.2.1.4 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 11–7](#) shows the **History** view for a BPEL file.

Figure 11–7 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.

11.2.1.5 Component Palette

Activities are the building blocks of the BPEL process service component. The **BPEL Activities** selection of the **Component Palette** shown in the upper right part of [Figure 11–1](#) displays a set of activities that you drag and drop into the **Design** window of the BPEL process service component. The **Component Palette** displays only 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 11–8](#) shows the **BPEL Activities** selection of the **Component Palette**. This list enables you to select activities to drag and drop into your BPEL process service component.

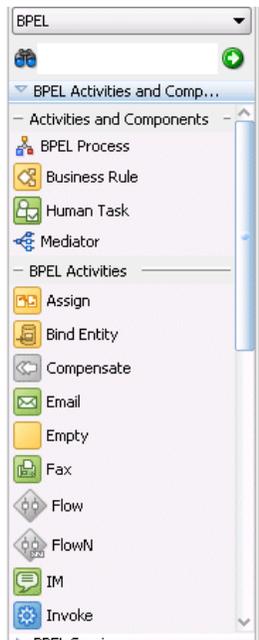
Figure 11–8 Component Palette - BPEL Activities

Figure 11–9 shows the **BPEL Services** selection of the **Component Palette**. This list enables you to drag and drop adapters, partner links, or decision services into your BPEL process service component.

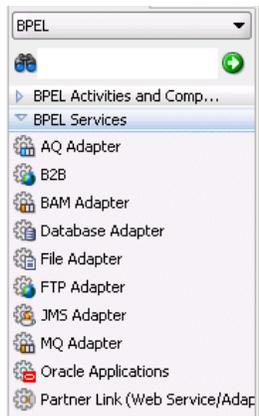
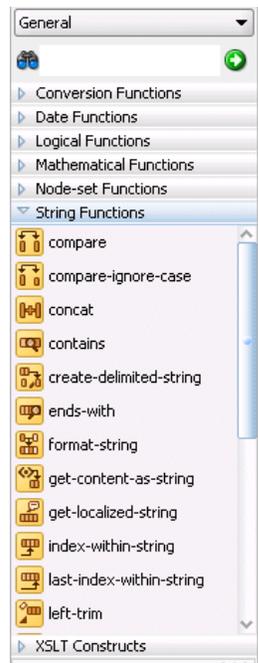
Figure 11–9 Component Palette - Services

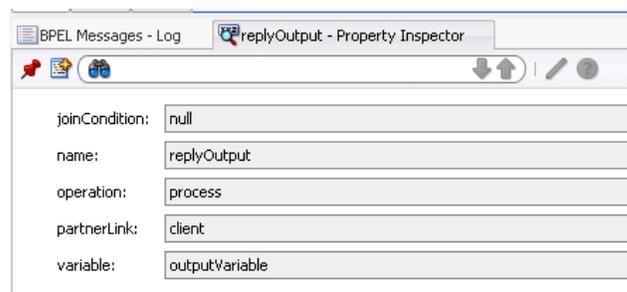
Figure 11–10 shows the **String Functions** category of the **Component Palette** that displays when you work in the transformation window of a **transform** activity.

Figure 11–10 Component Palette - Functions

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

11.2.1.6 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 11–11](#).

Figure 11–11 Property Inspector

11.2.1.7 Structure Window

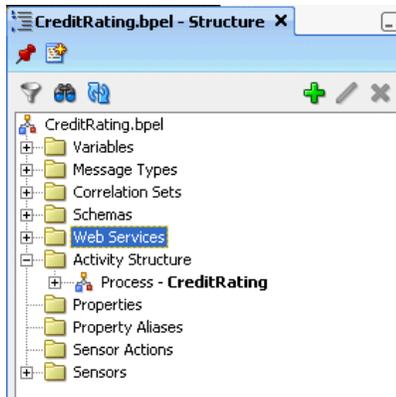
The **Structure Window** shown in the lower left part of [Figure 11–1](#) offers a structural view of the data in the BPEL process service component currently selected in the **Design** window. You can perform a variety of tasks from this section, including:

- Importing schemas
- Defining message types
- Managing (creating, editing, and deleting) elements such as variables, aliases, correlation sets, partner links, and sensors

- Editing activities in the BPEL process flow sequence that displays in the **Design** window

Figure 11–12 shows the **Structure Window**.

Figure 11–12 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.
-
-

11.2.1.8 Log Window

Figure 11–1 displays messages about the status of validation and compilation.

To ensure that a BPEL process service component validates correctly, you must ensure that the following information is correct:

- The BPEL process service component must have an input variable.
- A partner link must be selected.
- A partner role must be selected.
- The operation must not be empty.
- The input variable type must match the partner link operation type.

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.

11.3 Introduction to Activities

Activities are the building blocks of a BPEL process service component. Oracle JDeveloper includes a set of activities that you drag and drop into a BPEL process service component. You then double-click an activity to define its attributes (property values). Figure 11–4 on page 11-7 provides an example of this design process.

Activities enable you to perform specific tasks within a BPEL process service component. For example:

- An assign activity enables you to manipulate data, such as copying the contents of one variable to another.



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

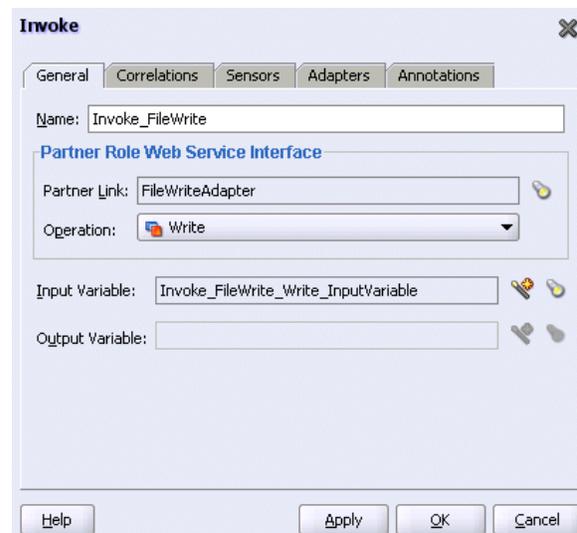


- A receive activity waits for an asynchronous callback response message from a service.



Figure 11–13 shows an example of a property window (for this example, an invoke activity). In this example, you invoke a partner link named **Invoke_FileWrite** and define its attributes.

Figure 11–13 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.

11.4 Introduction to Partner Links

The term *partner link* has also been mentioned frequently in this chapter. 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 JDeveloper. [Figure 11–14](#) shows the partner link icon (in this example, named **WriteRecord**).

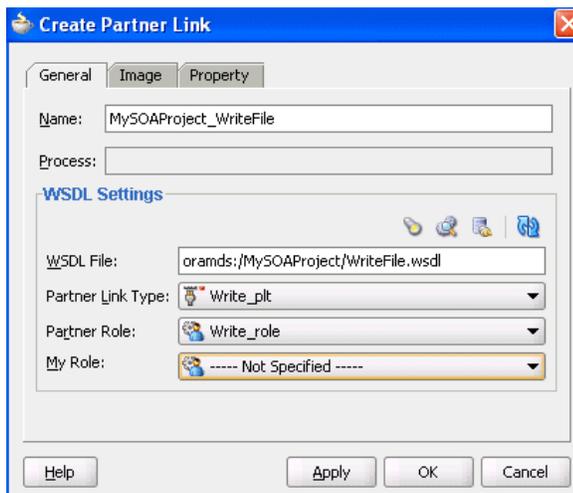
Figure 11–14 PartnerLink Icon



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 context of the conversation. [Figure 11–4](#) on page 11-7 shows an example of a partner link named **WriteDBRecord** being invoked by a BPEL process service component.

[Figure 11–15](#) shows an example of the attributes of a partner link for a service named **WriteFile**.

Figure 11–15 PartnerLink Window



[Table 11–4](#) describes the fields of the Create Partner Link window.

Table 11–4 Create Partner Link Window Fields

Field	Description
Name	A unique and recognizable name you provide for the partner link.
WSDL File	The name and location of the Web Services Description Language (WSDL) file that you select for the partner link. Click the SCA Service Explorer flashlight icon (second icon from the left above the WSDL File field) to access a window for selecting the WSDL file to use.
Partner Link Type	The partner link defined in the WSDL file.
Partner Type	The role performed by the partner link (in this example, the Write_role service).

Table 11–4 (Cont.) Create Partner Link Window Fields

Field	Description
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.

11.5 Partner Link Creation and the SOA Composite Editor

The method by which you create partner links within the BPEL process in Oracle JDeveloper impacts how the partner link displays above in the SOA Composite Editor. This section describes this impact.

This section contains the following topics:

- [Section 11.5.1, "Creating a Partner Link For an Outbound Adapter"](#)
- [Section 11.5.2, "Creating a Partner Link for an Inbound Adapter"](#)
- [Section 11.5.3, "Creating a Partner Link from an Abstract WSDL to Call a Service"](#)
- [Section 11.5.4, "Creating a Partner Link from an Abstract WSDL to Implement a Service"](#)
- [Section 11.5.5, "Creating a Human Task or Decision Service"](#)
- [Section 11.5.6, "Creating a Partner Link From an Existing Human Task, Decision Service, or Mediator Routing Service"](#)

See Also: [Chapter 2, "Introduction to the SOA Composite Editor"](#) for details about using the SOA Composite Editor

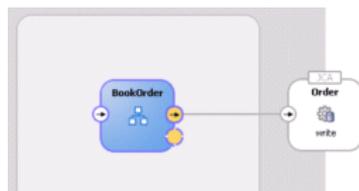
11.5.1 Creating a Partner Link For an Outbound Adapter

[Table 11–5](#) describes the impact on the SOA Composite Editor.

Table 11–5 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle JDeveloper...	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 11–16](#) shows how this method of creation appears in the SOA Composite Editor.

Figure 11–16 SOA Composite Editor Impact

11.5.2 Creating a Partner Link for an Inbound Adapter

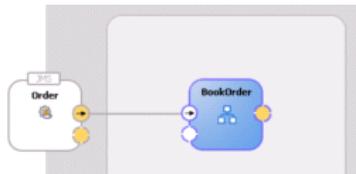
Table 11-6 describes the impact on the SOA Composite Editor.

Table 11-6 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle JDeveloper...	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 11-17 shows how this method of creation appears in the SOA Composite Editor.

Figure 11-17 SOA Composite Editor Impact



11.5.3 Creating a Partner Link from an Abstract WSDL to Call a Service

Table 11-7 describes the impact on the SOA Composite Editor.

Table 11-7 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle JDeveloper...	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

Figure 11-18 shows how this method of creation appears in the SOA Composite Editor.

Figure 11-18 SOA Composite Editor Impact



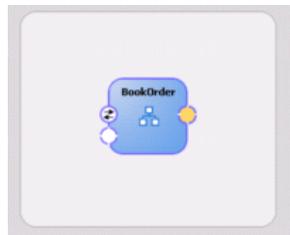
11.5.4 Creating a Partner Link from an Abstract WSDL to Implement a Service

Table 11-8 describes the impact on the SOA Composite Editor.

Table 11–8 *Impact of Partner Link Creation on the SOA Composite Editor*

Creating the Following for a BPEL Process in Oracle JDeveloper...	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 SOAP reference with the specified interface and callback interface already 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 11–19 shows how this method of creation appears in the SOA Composite Editor.

Figure 11–19 *SOA Composite Editor Impact*

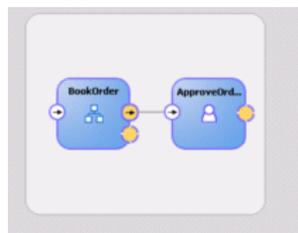
11.5.5 Creating a Human Task or Decision Service

Table 11–9 describes the impact on the SOA Composite Editor.

Table 11–9 *Impact of Partner Link Creation on the SOA Composite Editor*

Creating the Following for a BPEL Process in Oracle JDeveloper...	Displays the Following in the SOA Composite Editor...
A human task or decision service is created	<ul style="list-style-type: none"> ■ A human task or decision service 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 11–20 shows how this method of creation appears in the SOA Composite Editor.

Figure 11–20 *SOA Composite Editor Impact*

11.5.6 Creating a Partner Link From an Existing Human Task, Decision Service, or Mediator Routing Service

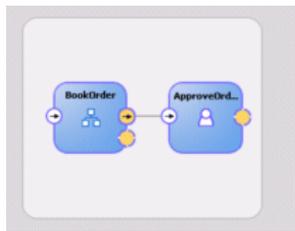
Table 11–10 describes the impact on the SOA Composite Editor.

Table 11–10 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle JDeveloper...	Displays the Following in the SOA Composite Editor...
A partner link by dragging an existing human task, decision service, or mediator routing 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, decision service, or mediator routing service

Figure 11–21 shows how this method of creation appears in the SOA Composite Editor.

Figure 11–21 SOA Composite Editor Impact



11.6 Introduction to Oracle BPEL Server

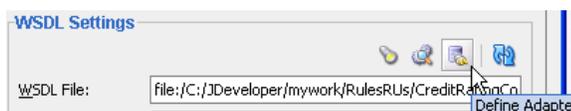
After you complete the design of the BPEL process service component, you compile and deploy the component to Oracle BPEL Server. If compilation and deployment are successful, you can run and manage the BPEL process service component from Oracle Enterprise Manager 11g Application Server Control Console.

Deploying the SOA project involves deploying the SOA composite application. When you deploy an application, a service assembly archive (SAR) file is created (for example, named `sca_OrderBooking_rev1.0.sar`). The SAR file consists of an EAR file and Oracle metadata.

11.7 Introduction to Oracle BPEL Process Manager Technology Adapters

The Web Service Window shown in Figure 11–15 on page 11-14 also enables you to take advantage of another key feature that Oracle BPEL Process Manager and Oracle JDeveloper provide. Click the **Define Adapter Service** icon shown in Figure 11–22 to access the Adapter Configuration wizard.

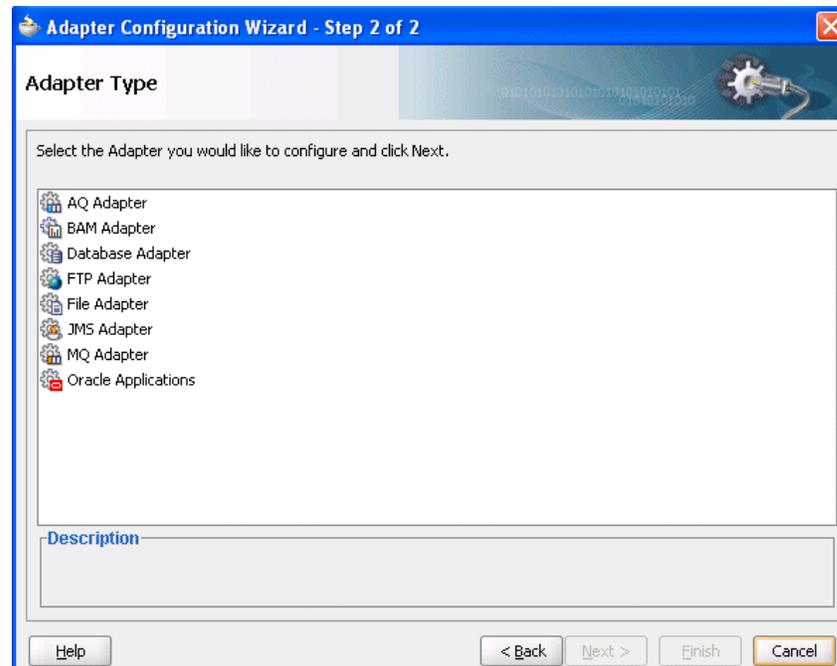
Figure 11–22 Defining an Adapter



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, Java Message Services (JMS), MQ, and Oracle E-Business Suite. This wizard enables you to configure the types of adapters shown in [Figure 11–23](#) for use with the BPEL process service component:

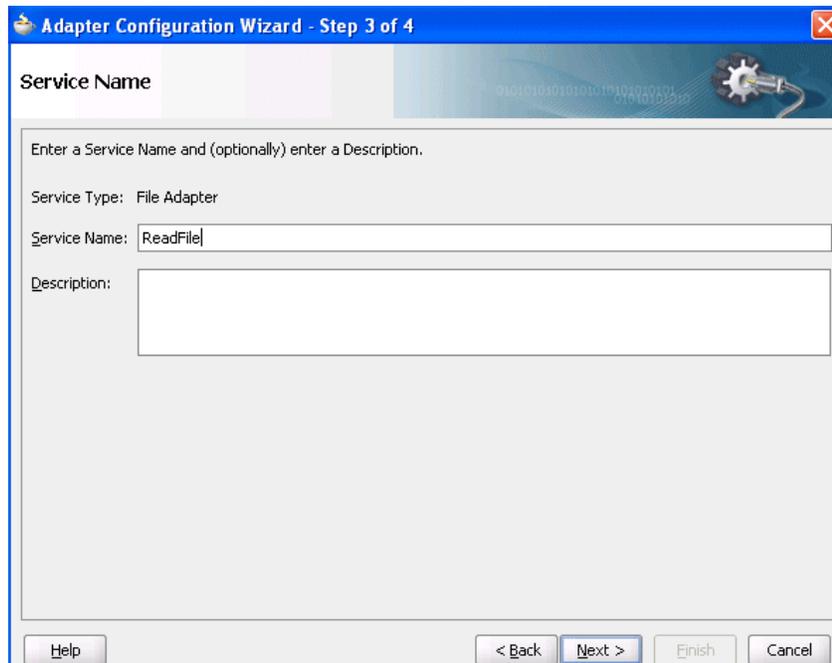
Figure 11–23 Adapter Types



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.
- 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))
- 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

When you select an adapter type, the Service Name window shown in [Figure 11–24](#) prompts you to enter a name. For this example, **File Adapter** was selected in [Figure 11–23](#). 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**). 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 11–24 Adapter Service Name

Adapter Configuration Wizard - Step 3 of 4

Service Name

Enter a Service Name and (optionally) enter a Description.

Service Type: File Adapter

Service Name: ReadFile

Description:

Help < Back Next > Finish Cancel

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.

See Also: [Chapter 2, "Introduction to the SOA Composite Editor"](#) for details about adding adapters to SOA composite applications

Manipulating XML Data in BPEL Processes

This chapter describes how to manipulate XML data in BPEL, including the use of XPath expressions.

This chapter includes the following sections:

- [Section 12.1, "Introduction to Manipulating XML Data in BPEL Concepts"](#)
- [Section 12.2, "Delegating XML Data Operations to Data Provider Services"](#)
- [Section 12.3, "Initializing a Variable with Expression Constants or Literal XML"](#)
- [Section 12.4, "Copying Between Variables"](#)
- [Section 12.5, "Accessing Fields Within Element-Based and Message Type-Based Variables"](#)
- [Section 12.6, "Assigning Numeric Values"](#)
- [Section 12.7, "Using Mathematical Calculations with XPath Standards"](#)
- [Section 12.8, "Assigning String Literals"](#)
- [Section 12.9, "Concatenating Strings"](#)
- [Section 12.10, "Assigning Boolean Values"](#)
- [Section 12.11, "Assigning a Date or Time"](#)
- [Section 12.12, "Manipulating Attributes"](#)
- [Section 12.13, "Manipulating XML Data with bpelex Extensions"](#)
- [Section 12.14, "Validating XML Data with bpelex:validate"](#)
- [Section 12.15, "Manipulating XML Data Sequences That Resemble Arrays"](#)
- [Section 12.16, "Converting from a String to an XML Element"](#)
- [Section 12.17, "Understanding the Differences Between Document-Style and RPC-Style WSDL Files"](#)
- [Section 12.18, "Manipulating SOAP Headers in BPEL"](#)
- [Section 12.19, "Use Cases for Manipulating XML Data in BPEL"](#)

12.1 Introduction to Manipulating XML Data in BPEL Concepts

This section covers the following topics:

- [Section 12.1.1, "XML Data in BPEL"](#)
- [Section 12.1.2, "Data Manipulation and XPath Standards"](#)

12.1.1 XML Data in BPEL

In a BPEL process service component, every piece of data is 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 WSDL file for the flow, the WSDL files for the services it invokes, or the XSD file referenced by those WSDL files. Therefore, all 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).

12.1.2 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 12-1](#) shows the formal syntax, as described in the *Business Process Execution Language for Web Services Specification*:

Example 12-1 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 12-2](#):

Example 12-2 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 window that includes a **From** section and a **To** section. This reflects the preceding BPEL source code syntax.

Rather than repeating all syntax details, this chapter shows and describes excerpts taken primarily from sample projects provided in the `SOA_Oracle_Home\bpel\samples\references` directory.

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 12-3](#) provides an example:

Example 12-3 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:** You can also create custom XPath functions. If you do, you must register them in the `ORACLE_HOME\j2ee\home\applications\soa-infra\configuration\bpel-xpath-functions-config.xml` file:

Then, copy the class files under the `oc4j\patch\classes` directory.

For more information about writing custom XPath functions, refer to: <http://www.oracle.com/technology/bpel>

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 12.13, "Manipulating XML Data with bpelx Extensions"](#) on page 12-16), or as a Web service. For more information on calling Java code from within BPEL, see the tutorial under the **BPEL Tutorials** link at <http://www.oracle.com/technology/bpel>. For XSLT, Oracle BPEL Process Manager includes XPath functions that execute these transformations.

For more information about XPath and XQuery transformation code examples, see the following:

- `SOA_Oracle_Home\bpel\samples\tutorials\114.XSLTTransformations`
- [Chapter 4, "XSLT Mapper and Transformations"](#)

The following sections show related definitions in the BPEL and WSDL files that help explain the examples.

12.2 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 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.

For this release, the entity variable can be used with an Application Development Framework (ADF) Business Component (ADF BC) data provider service using service data object (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 12–1](#) 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 12–1 Data Manipulation Capabilities in Previous and Current Release

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 ADF BC 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 the data is required. 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.
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 already 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 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 12–1](#) are still supported in release 11g.

12.2.1 How to Create an Entity Variable

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

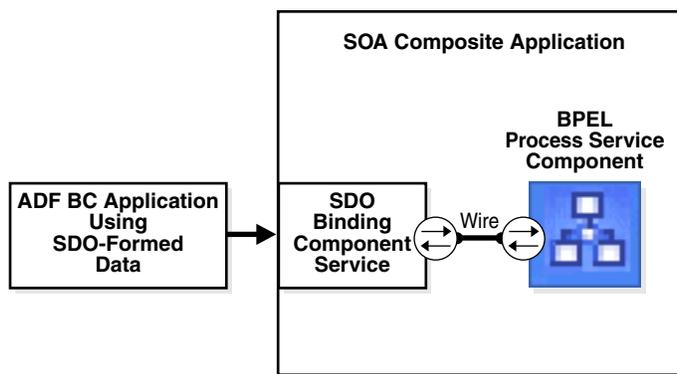
In previous releases of Oracle BPEL Process Manager, all variable data was in DOM form. With release 11g, variable data in SDO form is also supported. DOM and SDO variables in BPEL process service components are implicitly converted to the required forms. For example, an Oracle BPEL process service component 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 previously described, 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 ADF BC 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.

12.2.1.1 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 12-1](#).

Figure 12-1 Inbound Direction



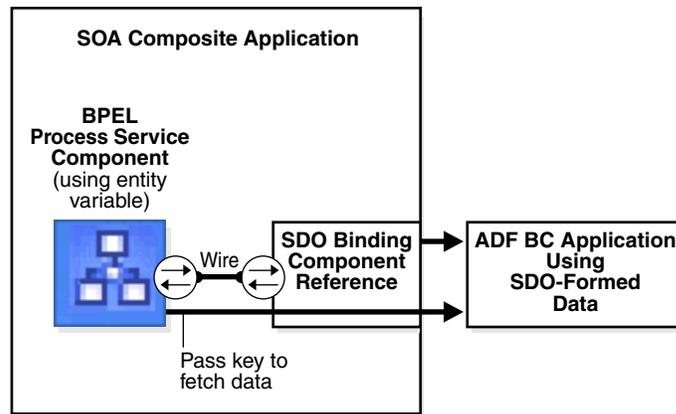
You use the SOA Composite Editor and Oracle JDeveloper to perform the following tasks:

- 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 together.
- Define the details of the BPEL process service component.

For more information about using the SOA Composite Editor, see [Chapter 2, "Introduction to the SOA Composite Editor"](#).

12.2.1.2 Understanding How SDO Works in the Outbound Direction

The SDO binding component reference enables messages to be sent from the composite application to ADF BC application external partners in the outside world, as shown in [Figure 12-2](#).

Figure 12-2 Outbound Direction

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

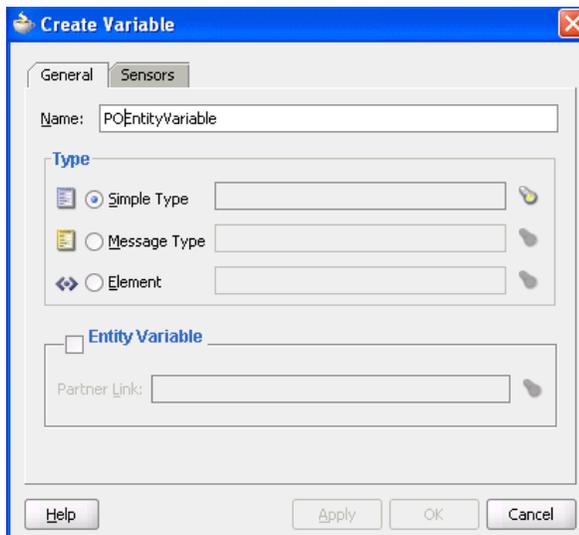
- The ADF BC application is deployed and automatically registered as an SDO service in the Service Infrastructure
- You use Oracle JDeveloper to browse for and discover this application as an SDO service and create a partner link connection
- The `composite.xml` file is automatically updated with reference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

For more information about the Service Infrastructure, see [Chapter 1, "Service-Oriented Architecture and Oracle SOA Suite"](#).

12.2.1.3 Creating an Entity Variable and Choosing a Partner Link

You now create an entity variable and select a partner link for the ADF BC application.

1. Go to the **Structure** window of the BPEL process service component in Oracle JDeveloper.
2. Right-click **Variables** and select **Expand All Child Nodes**.
3. Right-click the second **Variables** folder and select **Create Variable**.
The Create Variable window appears.
4. Enter a name in the Name field, as shown in [Figure 12-3](#).

Figure 12–3 Create Variable Dialog

5. Click the **Entity Variable** check box and select the **flashlight** icon to the right of the **Partner Link** field.

The Partner Link Chooser window appears with a list of available services, including the SDO service.

6. Browse for and select the service for the ADF BC application.
7. Click **OK** to close the Partner Link Chooser and Create Variable windows.

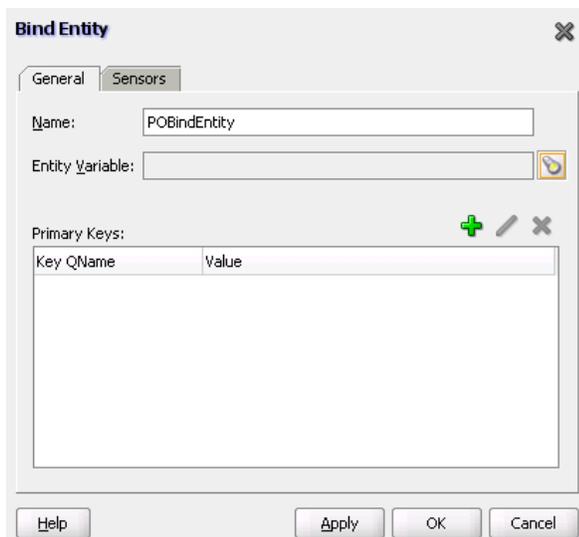
12.2.1.4 Creating a Binding Key

You now create a key to point to the data in the ADF BC data provider service.

1. Drag and drop a **Bind Entity** activity into your BPEL process service component.

The Bind Entity window appears.

2. Enter a name in the **Name** field, as shown in [Figure 12–4](#).

Figure 12–4 Bind Entity Dialog

3. Click the **flashlight** icon to the right of the **Entity Variable** field.
The Variable Chooser window appears.
4. Select the entity variable created in [Section 12.2.1.3, "Creating an Entity Variable and Choosing a Partner Link"](#) on page 12-7 and click **OK**.
5. Click the + sign in the **Unique Keys** section.
The Create Key window appears.
6. Enter the following details to define the binding key:

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

7. Click **OK** to close the Create Key window.
A name-pair value appears in the **Unique Keys** table. Design is now complete.
8. Click **OK** to close the Bind Entity window.
After the Bind Entity activity is executed at runtime, the entity variable is ready to be used.

12.3 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.

12.3.1 How To Assign a Literal XML Element

[Example 12-4](#) assigns a literal `result` element to the `payload` part of the output variable:

Example 12-4 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>
      </result>
    </from>
  </copy>
</assign>
```

```

        <quantity>0</quantity>
        <approved/>
        <message/>
    </result>
</from>
    <to variable="output" part="payload"/>
</copy>
</assign>

```

For more information, see the *SOA_Oracle_Home\bpel\samples\references\Assign* sample.

12.4 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.

12.4.1 How to Copy Between Variables

[Example 12–5](#) 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 12–5 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 12–6](#):

Example 12–6 Variable Definition

```

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

```

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

Example 12–7 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*.

12.5 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, which in turn uses XML schema complex types. To do this, you specify an XPath query in the `from` or `to` clause of the `assign` activity.

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

In [Example 12-8](#), 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 12-8 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 12-9](#) shows how the BPEL file defines the variables involved in this assignment:

Example 12-9 BPEL File Definition

```
<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 `CreditFlowService.wsdl` as shown in [Example 12-10](#):

Example 12-10 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 `CreditRatingService.wsdl` as shown in [Example 12-11](#):

Example 12-11 CreditRatingServiceRequestMessage Definition

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

For more information, see the `SOA_Oracle_Home\bpel\samples\utils\CreditRatingService` sample.

12.6 Assigning Numeric Values

You can assign numeric values in XPath expressions.

12.6.1 How to Assign Numeric Values

[Example 12–12](#) shows how to assign an XPath expression with the integer value 100.

Example 12–12 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>
```

For more information, see the *SOA_Oracle_Home\bpel\samples\references\Assign* sample.

12.7 Using Mathematical Calculations with XPath Standards

You can use simple mathematical expressions like the one in [Section 12.7.1, "How To Use Mathematical Calculations with XPath Standards,"](#) which increments a numeric value.

12.7.1 How To Use Mathematical Calculations with XPath Standards

In [Example 12–13](#), 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 12–13 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 12–14](#):

Example 12–14 \$variable Syntax Use

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

For more information, see the *SOA_Oracle_Home\bpel\samples\references\Assign* sample.

12.8 Assigning String Literals

You can assign string literals to a variable in BPEL.

12.8.1 How to Assign String Literals

The code in [Example 12–15](#) 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 12–15 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>
```

For more information, see the *SOA_Oracle_Home\bpel\samples\references\Assign* sample.

12.9 Concatenating Strings

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

12.9.1 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 12–16](#), `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 12–16 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>
```

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

For more information, see the *SOA_Oracle_Home\bpel\samples\references\Assign* sample.

12.10 Assigning Boolean Values

You can assign Boolean values with the XPath Boolean function.

12.10.1 How to Assign Boolean Values

[Example 12–17](#) 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 12–17 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. This means 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.

For more information, see the *SOA_Oracle_Home\bpel\samples\references\Assign* sample.

12.11 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*.

12.11.1 How to Assign a Date or Time

[Example 12–18](#) shows an example that uses the function `getCurrentDate`.

Example 12–18 Date or Time Assignment

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

In [Example 12–19](#), 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 12–19 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>

```

For more information, see the *SOA_Oracle_Home\bpel\samples\references\XPathFunction* sample.

12.12 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.

12.12.1 How to Manipulate Attributes

The code in [Example 12–20](#) fetches and copies the `custId` attribute from this XML data:

Example 12–20 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 12–21](#) selects the `custId` attribute of the customer field and assigns it to the variable `custId`:

Example 12–21 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 12–22](#):

Example 12–22 custId Attribute Definiton

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

For more information, see the *SOA_Oracle_Home\bpel\samples\references\XPath* sample.

12.13 Manipulating XML Data with bpelx Extensions

You may want to perform various operations on XML data in assign activities. The following bpelx extension types provide this functionality:

- [Section 12.13.1, "How to Use bpelx:append"](#)
- [Section 12.13.2, "How to Use bpelx:insertBefore"](#)
- [Section 12.13.3, "How to Use bpelx:insertAfter"](#)
- [Section 12.13.4, "How to Use bpelx:remove"](#)
- [Section 12.13.5, "How to Use bpelx:rename and XSD Type Casting"](#)
- [Section 12.13.6, "How to Use bpelx:copyList"](#)

12.13.1 How to Use bpelx:append

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 12–23](#) provides an example.

Example 12–23 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 12–24](#) consolidates multiple bills of material into one single bill of material by appending multiple `b:part`'s for one BOM to `b:parts` of the consolidated BOM.

Example 12–24 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>

```

12.13.2 How to Use bpelx:insertBefore

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 12–25](#) provides an example.

Example 12–25 `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 more than one node is 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 12–26](#) shows the syntax before the execution of `<insertBefore>`. The value of `addrVar` is:

Example 12–26 Presyntax Execution

```

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

```

After the execution of the syntax in [Example 12–27](#) in the BPEL process service component file:

Example 12–27 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 12–28](#) shows the value of `addrVar`:

Example 12–28 `addrVar` Value

```

<a:usAddress>

```

```

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

```

12.13.3 How to Use bpelx:insertAfter

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 12–29](#) provides an example.

Example 12–29 *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 12.13.2, "How to Use bpelx:insertBefore"](#) on page 12-17, 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 12–30](#) shows the syntax before the execution of `<insertAfter>`. The value of `addrVar` is:

Example 12–30 *Presyntax Execution*

```

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

```

After execution of the syntax shown in [Example 12–31](#) in the BPEL process service component file:

Example 12–31 *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 12–32](#) shows the value of `addrVar`:

Example 12–32 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.

12.13.4 How to Use bpelx:remove

The `bpelx:remove` extension in an assign activity enables a BPEL process service component to remove a variable. [Example 12–33](#) provides an example.

Example 12–33 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 12–34](#) shows `addrVar` with the value:

Example 12–34 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 12–35](#) in the BPEL process service component file, the second address line of `Mailstop` is removed:

Example 12–35 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 12–36](#) in the BPEL process service component file, both address lines are removed:

Example 12–36 Removal of Both Address Lines

```

<bpel:assign>
  <bpelx:remove>
    <target variable="addrVar"
      query="/a:usAddress/a:addressLine" />
  </bpelx:remove>
</bpel:assign>

```

12.13.5 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 12–37](#) provides an example.

Example 12–37 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 or more element nodes. Otherwise, a `bpel:selectionFailure` fault is generated. The element nodes specified in the `from-spec` are renamed 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 12–38](#):

Example 12–38 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 12–39](#):

Example 12–39 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>

```

```

    </bpelx:rename>
</bpel:assign>

```

After executing the above casting (renaming), the data looks as shown in [Example 12–40](#) with `xsi:type` info added to Peter Smith:

Example 12–40 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 12–41](#) provides an example.

Example 12–41 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 12–42](#):

Example 12–42 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>

```

12.13.6 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 12–43](#) provides an example.

Example 12–43 *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. Then, the attributes returned by the `from-spec` query are appended to the parent element.

12.14 Validating XML Data with `bpelx:validate`

The `bpelx:validate` function enables you to verify code and identify invalid XML data.

12.14.1 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" />
```

If you want to verify the validity of XML data, set the `validateXML` property to `true` in `oc4j\j2ee\oc4j_soa\applications\soa-infra\configuration\bpel-config.xml`.

12.15 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 of more 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/bpel>.

Each of the following sections describes a particular requirement for data sequence manipulation. For a code example that describes all data sequences, see `ArraySample.bpel`, which takes a data sequence as input and loops through it, adding together individual line items in each data sequence element into a total value.

For more information, see the `ArraySample.bpel` sample file located at `SOA_Oracle_Home\bpel\samples\tutorials\112.Arrays`.

12.15.1 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 12-44](#), `addresses[1]` selects the first element of the `addresses` data sequence:

Example 12-44 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 12–45](#) 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 12–45 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.

For more information, see the *SOA_Oracle_Home\bpel\samples\references\XPath* sample.

12.15.2 How to Determine Sequence Size

If you need to know the run-time 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 12–46](#) calculates the number of elements in the `item` sequence and assigns it to the integer variable `lineItemSize`:

Example 12–46 Sequence Size Determination

```
<assign>
  <copy>
    <from expression="count(bpws:getVariableData('outpoint', 'payload',
      '/p:invoice/p:lineItems/p:item'))"/>
    <to variable="lineItemSize"/>
  </copy>
</assign>
```

For more information, see the *SOA_Oracle_Home\bpel\samples\references\XPathFunction* sample.

12.15.3 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 need to get the *n*th node out of a sequence, where the value of *n* is defined at run time. This section covers the following methods for dynamically indexing by applying a trailing XPath into expressions:

- [Section 12.15.3.1, "Applying a Trailing XPath to the Result of getVariableData"](#)
- [Section 12.15.3.2, "Using the bpel:append Extension to Append New Items to a Sequence"](#)

- [Section 12.15.3.3, "Merging Data Sequences"](#)
- [Section 12.15.3.4, "Generating Functionality Equivalent to an Array of an Empty Element"](#)

12.15.3.1 Applying a Trailing XPath to the Result of `getVariableData`

The dynamic indexing method shown in [Example 12–47](#) 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 12–47 *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 run time 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 already the root element. Specifying the root element name again in the XPath is redundant and is incorrect according to standard XPath semantics.

12.15.3.2 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 12–48](#) provides an example.

Example 12–48 *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.

For more information, see the *SOA_Oracle_Home\bpel\samples\tutorials\126.DataAggregator\AggregationTutorial* sample.

12.15.3.3 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 12–49](#) under `assign` demonstrate how to merge data sequences:

Example 12–49 *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>
    
```

For more information, see the *ArraySample.bpel* sample file located at *SOA_Oracle_Home\bpel\samples\tutorials\112.Arrays*.

12.15.3.4 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 12-50](#) shows an `append` statement initializing a purchase order (PO) document with 10 empty `<lineItem>` elements under `po`:

Example 12-50 `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.

12.15.4 What You May Need to Know About SOAP-Encoded Arrays

Oracle BPEL Process Manager provides limited support for SOAP-encoded arrays (`soapenc:arrayType`).

Consider one of the following methodologies to deal with SOAP arrays:

- A wrapper can be placed 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 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.

12.16 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 document object model (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.

12.16.1 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 12-51](#) provides an example:

Example 12-51 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>
```

For more information, see the *SOA_Oracle_Home\bpel\samples\references\XPathFunction* sample.

12.17 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 in [Example 12-52](#):

Example 12-52 XML Schema type 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 in [Example 12-53](#):

Example 12–53 XML Schema type Definition

```
<message name="LoanFlowRequestMessage">
  <part name="payload" type="s1:LoanApplicationType"/>
</message>
```

12.17.1 How To Use RPC-Style Files

This affects 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 the previous example). In document-style, the top-level node is the element name (for example, loanApplication).

[Example 12–54](#) and [Example 12–55](#) show what an XPath query string looks like if the LoanServices used in BPEL demo applications (such as LoanFlow) were RPC style.

Example 12–54 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 12–55 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>
```

For more information, see the following samples:

- SOA_Oracle_Home\bpel\samples\utils\AsyncLoanService (LoanServices)
- SOA_Oracle_Home\bpel\samples\demos\LoanDemo\LoanFlow (BPEL demo application)

12.18 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 more than one message 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 follows:

Example 12–56 `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 ..." .../>
```

12.18.1 How to Receive SOAP Headers in BPEL

This section provides an example of how to create BPEL and WSDL files to receive SOAP headers.

1. Create a WSDL file that declares header messages and the SOAP binding that binds them to the SOAP request. [Example 12–57](#) provides an example.

Example 12–57 WSDL File Contents

```
<message name="MessageIDHeader">
  <part name="MessageID" element="wsa:MessageID" />
</message>
<message name="ReplyToHeader">
  <part name="ReplyTo" element="wsa:ReplyTo" />
</message>

<!-- 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:ReplyToHeader" part="ReplyTo"
        use="literal" />
      <soap:header message="tns:MessageIDHeader" part="MessageID"
        use="literal" />
      <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.

Example 12–58 *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" />
  <variable name="messageID"
    messageType="tns:MessageIDHeader" />
  <variable name="replyTo"
    messageType="tns:ReplyToHeader" />
</variables>

<sequence>
  <!-- receive input from requestor -->
  <receive name="receiveInput" partnerLink="client"
    portType="tns:HeaderService" operation="initiate"
    variable="input"
    bpelx:headerVariable="customHeader messageID replyTo"
    createInstance="yes" />

```

12.18.2 How to Send SOAP Headers in BPEL

This section provides an example of how to send SOAP headers.

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`.

Example 12–59 *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" />

```

12.19 Use Cases for Manipulating XML Data in BPEL

This chapter covers a variety of use cases for manipulating XML data. Topics include how to work with variables, sequences, and arrays, and how to perform tasks such as mathematical calculations. The explanations are largely by example, and provide an introduction to the supported specifications.

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 12.17, "Understanding the Differences Between Document-Style and RPC-Style WSDL Files"](#) on page 12-28.

Note: The `103.XMLDocuments` sample is not included in the `soa-samples.zip` file for beta 3.

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 create a partner link and invoke activity and set up the components necessary to perform a synchronous callback. This chapter also examines how these components are coded.

This chapter includes the following sections:

- [Section 13.1, "Introduction to Invoking a Synchronous Web Services"](#)
- [Section 13.2, "Invoking a Synchronous Web Service"](#)
- [Section 13.3, "Use Case for Invoking a Synchronous Web Service"](#)

13.1 Introduction to Invoking a Synchronous Web Services

Synchronous Web services provide an immediate response to a query. BPEL can connect to synchronous Web services through a partner link, send data, and then receive the reply using a synchronous callback.

A synchronous callback requires the following components:

- **Partner link:** Defines the location and the role of the Web services that the BPEL process service component connects to in order to perform tasks, as well as 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.
- **Invoke activity:** Opens a port in the BPEL process service component to send and receive data. It uses this port to submit the required data and receive the response. In the credit rating service example, the invoke activity submits the stock code entered by the customer to the stock quote service and receives a stock quote in return. For synchronous callbacks, only one port is needed for both the send and receive functions.

You can set timeout values with the attribute `syncMaxWaitTime` in the `SOA_ORACLE_HOME\j2ee\home\applications\soa-infra\configuration\bpel-config.xml` file. If the BPEL process service component does not receive a reply within the specified time, then the activity fails.

For more information about `syncMaxWaitTime`, see *Oracle Application Server Performance Guide*.

13.2 Invoking a Synchronous Web Service

This section examines a synchronous callback operation using the `QuoteConsumer.bpel` file. For a more step-by-step approach, see <http://www.oracle.com/technology/bpel> and download the files under **Training Material**.

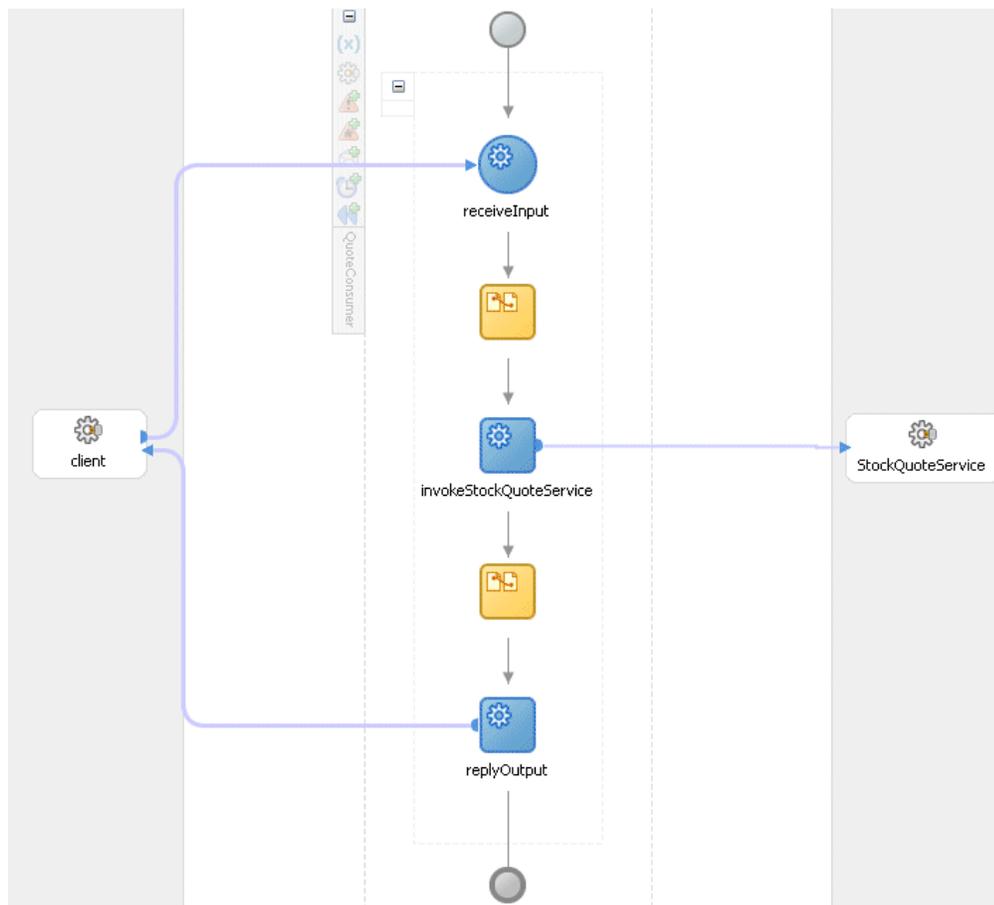
13.2.1 How to Invoke a Synchronous Web Service

To invoke a synchronous web service:

1. In the Component Palette, drag and the necessary partner link, invoke activity, and assign activities into the designer.
2. Edit their dialogs.

Figure 13–1 shows the diagram of the `QuoteConsumer.bpel` file, which defines a simple application with five activities.

Figure 13–1 Diagram of `QuoteConsumer.bpel`



The following actions take place (in order of priority):

1. The **receiveInput** **receive** activity receives input from the user (client), as defined in the `QuoteConsumer.wsdl` file.
2. The first **assign** activity packages the data from the client so that it can be accepted by the **invokeStockQuote** service.

3. The **invokeStockQuoteService** activity sends the repackaged data to the **StockQuoteService** service and receives a response.
4. A second **assign** activity repackages this response into a **replyOutput** activity so that it can be accepted by the client application.
5. The **replyOutput** activity sends the repackaged response back to the client.

13.2.2 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 WSDL files.

13.2.2.1 Partner Link in the BPEL Code

In the 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 `StockQuoteService` partner link definition is shown in [Example 13-1](#):

Example 13-1 Partner Link Definition

```
<partnerLinks>
  <!--
    The 'client' role represents the requester of this service. It is
    used for callback. The location and correlation information associated
    with the client role are automatically set using WS-Addressing.
  -->
  <partnerLink name="client" partnerLinkType="samples:QuoteConsumer"
    myRole="QuoteConsumerProvider" partnerRole="QuoteConsumerRequester"/>
  <partnerLink
name="StockQuoteService"partnerLinkType="services:StockQuoteService"
  partnerRole="StockQuoteServiceProvider"/>
</partnerLinks>
```

Following the partner link are global variable definitions that are accessible throughout the BPEL process service component. The types for these variables are defined in the WSDL for the process itself. [Example 13-2](#) provides an example.

Example 13-2 Variable Definitions

```
<variables>
  <!-- Reference to the message passed as input during initiation -->
  <variable name="input" messageType="tns:QuoteConsumerRequestMessage"/>
  <!-- Reference to the message that will be sent back to the
    requestor during callback
  -->
  <variable name="output" messageType="tns:QuoteConsumerResultMessage"/>
  <variable name="request" messageType="services:StockQuoteServiceRequest"/>
  <variable name="response" messageType="services:StockQuoteServiceResponse"/>
</variables>
```

The WSDL file defines the interface to your BPEL process service component—the messages that it accepts and returns, operations that are supported, and other parameters.

13.2.2.2 Partner Link Type and Port Type in the BPEL Code

The Web service's `QuoteConsumer.wsdl` file contains two sections that enable the web service to work with BPEL process service components:

- [Section 13.2.2.2.1, "partnerLinkType Section of the QuoteConsumer.wsdl File"](#)
- [Section 13.2.2.2.2, "portType Section of the QuoteConsumer.wsdl File"](#)

13.2.2.2.1 partnerLinkType Section of the QuoteConsumer.wsdl File The `partnerLinkType` section of the `QuoteConsumer.wsdl` file defines the following characteristics of the conversion between a BPEL process service component and the loan application approver Web service:

- The role (operation) played by each
- The `portType` provided by each for receiving messages within the context of the conversation

[Example 13–3](#) provides an example:

Example 13–3 *partnerLinkType Definition*

```
<!--
PartnerLinkType definition
-->
<!-- the QuoteConsumer partnerLinkType binds the service and
requestor portType into an asynchronous conversation.
-->
<plnk:partnerLinkType name="QuoteConsumer">
  <plnk:role name="QuoteConsumerProvider">
    <plnk:portType name="tns:QuoteConsumer" />
  </plnk:role>
  <plnk:role name="QuoteConsumerRequester">
    <plnk:portType name="tns:QuoteConsumerCallback" />
  </plnk:role>
</plnk:partnerLinkType>
```

13.2.2.2.2 portType Section of the QuoteConsumer.wsdl File A port type is a collection of related operations implemented by a participant in a conversation. A port type defines what information is passed back and forth, the form of that information, and so forth. A synchronous callback requires only one port type that both sends a request and receives the response, while an asynchronous callback (one where the reply is not immediate) requires two port types, one to send the request, and another to receive the reply when it arrives.

[Example 13–4](#) shows the `portType` section of the `QuoteConsumer.wsdl` file. This is the stock quote Web service to which the client submits the stock code that the customer has entered.

Example 13–4 *portType Definition*

```
<!--
PortType definition
-->

<!-- portType implemented by the QuoteConsumer BPEL process -->
<portType name="QuoteConsumer">
  <operation name="initiate">
    <input message="tns:QuoteConsumerRequestMessage" />
  </operation>
</portType>
```

```

<!-- portType implemented by the requester of QuoteConsumer BPEL process
      for asynchronous callback purposes
-->
<portType name="QuoteConsumerCallback">
  <operation name="onResult">
    <input message="tns:QuoteConsumerResultMessage"/>
  </operation>
</portType>

```

Synchronous services have one port type. The port initiates the synchronous process and calls back the client with the response. In this example, the `portType` `CreditRatingService` receives the stock code and returns the stock quote.

13.2.2.3 Invoke Activity for Performing a Request

The `invoke` activity includes the `request` global input variable defined in the `variables` section. The credit rating Web service uses this `request` global input variable. This variable contains the customer's social security number. The response variable contains the credit rating returned by the credit rating service. [Example 13–5](#) provides an example.

Example 13–5 Invoke Activity

```

<sequence>
<!-- Receive input from requestor. Note: This maps to operation defined in
QuoteConsumer.wsdl
-->
<receive name="receiveInput" partnerLink="client" portType="samples:QuoteConsumer"
operation="initiate" variable="input" createInstance="yes"/>
<assign>
<copy>
<from variable="input" part="payload" query="/tns:symbol"/>
<to variable="request" part="symbol" query="/symbol"/>
</copy>
</assign>
<!-- Generate content of output message based on the content of the input message.
-->
<invoke name="invokeStockQuoteService" partnerLink="StockQuoteService"/>
<assign>
<copy>
<from variable="response" part="result" query="/result"/>
<to variable="output" part="payload" query="/tns:result"/>
</copy>
</assign>
<!-- Asynchronous callback to the requester. Note: the callback location and
correlation id is transparently handled using WS-addressing. -->
<invoke name="replyOutput" partnerLink="client"
portType="samples:QuoteConsumerCallback" operation="onResult"
inputVariable="output"/>
</sequence>

```

13.2.2.4 Synchronous Callback in BPEL Code

The BPEL code shown in [Example 13–6](#) performs the synchronous callback:

Example 13–6 Synchronous Callback

```

<assign>
  <copy>

```

```
<from variable="input" part="payload" query="/tns:symbol"/>
  <to variable="request" part="symbol" query="/symbol"/>
</copy>
</assign>
<invoke name="invokeStockQuoteService" partnerLink="StockQuoteService"
  portType="services:StockQuoteService" operation="process"
  inputVariable="request" outputVariable="response"/>

<!-- Generate content of output message based on the content of the input message.
-->
  <assign>
    <copy>
      <from variable="response" part="result" query="/result"/>
      <to variable="output" part="payload" query="/tns:result"/>
    </copy>
  </assign>
```

13.3 Use Case for Invoking a Synchronous Web Service

Note: The sample demonstrated in 104.SyncQuoteConsumer is not included in the `soa-samples.zip` file for beta 3.

Using synchronous Web services is demonstrated in 104.SyncQuoteConsumer. This sample shows a BPEL process service component sending a stock code to a Web service and receiving a stock quote in return. It examines how synchronous functionality is defined in the stock quote Web service's `CreditRatingService.wsdl` file (the Web service to be called) and the client's `QuoteConsumer.bpel` file and `composite.xml` deployment description file.

For more information, see the BPEL and WSDL files located in the `SOA_Oracle_Home\bpel\samples\tutorials\104.SyncQuoteConsumer\bpel` directory.

Invoking an Asynchronous Web Service from a BPEL Process

This chapter describes how to call an asynchronous Web service. Asynchronous messaging styles are very 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:

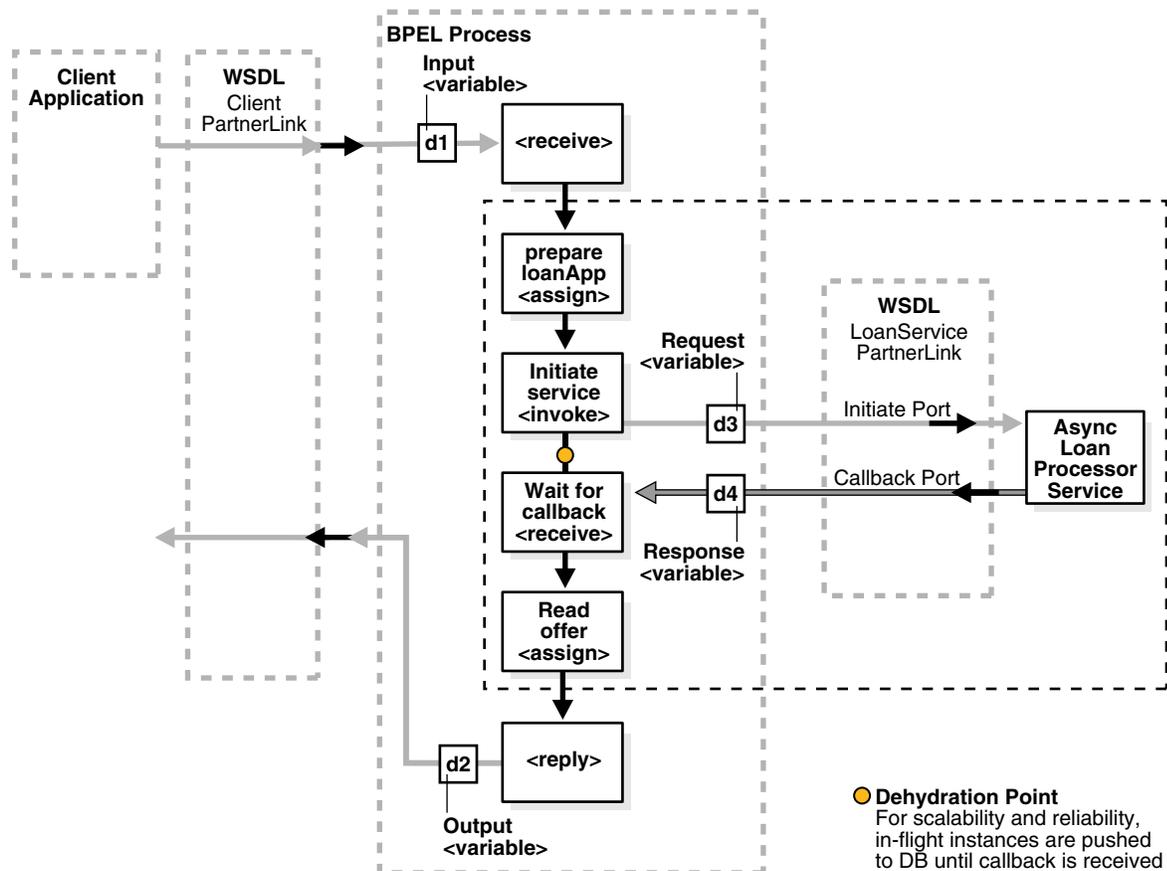
- Section 14.1, "Introduction to Invoking an Asynchronous Web Service"
- Section 14.2, "Invoking an Asynchronous Web Service"
- Section 14.3, "Using WS-Addressing in an Asynchronous Service"
- Section 14.4, "Using Correlation Sets in an Asynchronous Service"
- Section 14.5, "Use Case for Invoking a Asynchronous Web Service"

14.1 Introduction to Invoking an Asynchronous Web Service

This section examines how asynchronous functionality is defined in the loan application approver Web service's `LoanService.wsdl` file (the Web service to be called) and the client's `LoanBroker.bpel` file and `composite.xml` file.

[Figure 14-1](#) provides an overview of how this BPEL process service component works with the asynchronous loan processor web service.

Figure 14–1 Asynchronous Service Invocation



For the asynchronous Web service, which is indicated within the dotted rectangle between the BPEL process service component's receive and reply activities, the following actions take place:

1. An **assign** activity (**prepare LoanApp**) prepares the loan application.
2. An **invoke** activity (**initiate service**) 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 (**Wait for callback**), which has been tracked by the correlation ID.
4. An **assign** activity (**Read offer**) reads the loan application offer.

The remaining sections in this chapter provide specific details about the asynchronous functionality shown in [Figure 14–1](#).

For more information about asynchronous services, see the following files:

- `SOA_Oracle_Home\bpel\samples\utils\AsyncLoanService\LoanService.wsdl`

- *SOA_Oracle_*
Home\bpel\samples\tutorials\105.AsyncCompositeLoanBroker\bpel
\LoanBroker.bpel

14.2 Invoking an Asynchronous Web Service

To add asynchronous functionality to a BPEL process service component, complete the tasks in this section:

- [Section 14.2.1.1, "Step 1: Adding a Partner Link for an Asynchronous Service"](#)
- [Section 14.2.1.2, "Step 2: Adding an Invoke Activity"](#)
- [Section 14.2.1.3, "Step 3: Adding a Receive Activity"](#)
- [Section 14.2.1.4, "Step 4: Performing Additional Activities"](#)

14.2.1 How to Invoke an Asynchronous Web Service

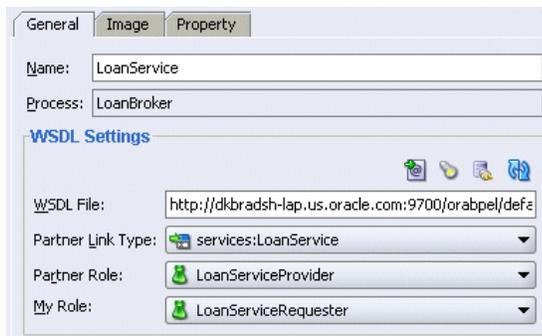
14.2.1.1 Step 1: Adding a Partner Link for an Asynchronous Service

These instructions describe how to create a partner link named LoanService for the loan application approver Web service.

1. Double-click the LoanBroker BPEL process service component in the composite application.
2. Expand **BPEL Services** in the **Component Palette**.
3. Drag and drop a **Partner Link** into one of the swim lanes.
The Create Web Service window appears.
4. Enter the following details to create a partner and select the loan application approver Web service:

- **Name:** Enter a name for the partner link.
- **Process:** The BPEL process service component name
- **WSDL File:** Enter the name of the WSDL file to use. Click the **SCA 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.
- **Partner Role:** Refers to the role of the external source, for example, provider. Select from the list.
- **My Role:** Refers to role of the BPEL process service component in this interaction, for example, requester. Select from the list.

[Figure 14–2](#) shows the Create Web Service dialog.

Figure 14–2 Create Web Service Dialog

5. Click **OK**.

A new partner link for the loan application approver Web service (United Loan) appears in the **Services** area of the `.bpel` file's diagram window.

For more information about partner links, see the following:

- [Section 14.2.2.1, "portType Section of the LoanService.wsdl File"](#)
- [Section 14.2.2.2, "partnerLinkType Section of the LoanService.wsdl File"](#)

14.2.1.2 Step 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 (LoanService). The loan application approver Web service uses the request input variable to receive the loan request from the client.

1. Expand **BPEL Activities** in the **Component Palette**.
2. Drag an **invoke** activity from the **Component Palette** to beneath the **receive** activity.
3. Go to the **Structure** window.
4. Right-click **Variables** and select **Expand All Child Nodes**.
5. Right-click the second **Variables** folder in the navigation tree, and click **Create Variable**.

The Create Variable dialog box appears.

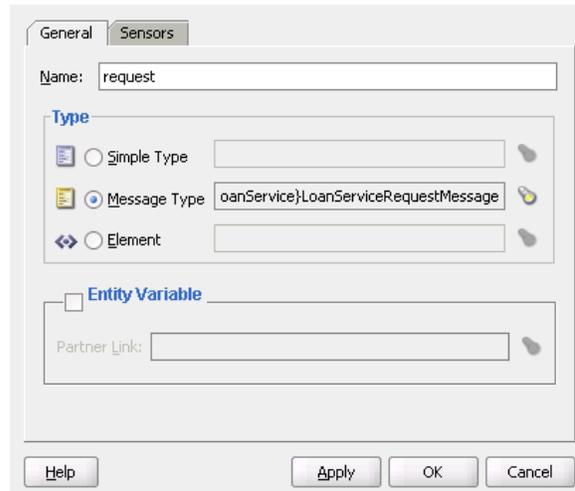
6. Enter the variable name and select **Message Type** from the options provided:
 - **Simple Type:** This option lets you select an XML schema simple type, for example, string, Boolean, and so on.
 - **Message Type:** This option enables you to select a WSDL message file definition of a partner link 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 flashlight icon to display the Type Chooser window. From here, expand the Message Types navigation window to select **Message Types > Partner Links > Loan Service > LoanService.wsdl > Message Types > LoanServiceRequestMessage**.

- **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 14–3 shows the Create Variable dialog.

Figure 14–3 Create Variable Dialog



7. Click **OK**.
8. Double-click the **invoke** activity to display the Invoke window.
9. In the Invoke window, select the **LoanService** partner link from the **Partner Link** list and **initiate** from the **Operation** list.
10. Select the input variable you created in Step 6, by clicking the second icon to the right of the **Input Variable** field.

The Variable Chooser window 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 14.2.2.5, "Invoke and Receive Activities."](#)

11. Click **OK**.

14.2.1.3 Step 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.

1. From the **Component Palette**, drag a **receive** activity to the location right after the **invoke** activity you created in [Section 14.2.1.2, "Step 2: Adding an Invoke Activity."](#)
2. Create a variable to hold the receive information by invoking the Create Variable window, as you did in Step 3 through Step 7, starting on page 14-4.

Figure 14–4 shows the Create Variable dialog.

Figure 14–4 Create Variable Dialog

3. Double-click the **receive** activity and change its name to **receive_invoke**.
4. Select **LoanService** from the **Partner Link** list and **onResult** from the **Operation** list. Do not select the **Create Instance** check box.
5. Select the variable you created in Step 3 through Step 7, starting on page 14-4.
6. Click **OK**.

The **receive** activity and the output variable are created. Because the initial **receive** activity in the **LoanBroker.bpel** file created the initial BPEL process service component instance, a second instance does not need to be created.

14.2.1.4 Step 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.

14.2.2 What Happens When You Invoke an Asynchronous Web Service

14.2.2.1 portType Section of the LoanService.wsdl File

The `portType` section of the `LoanService.wsdl` file defines the ports to be used for the asynchronous service.

Asynchronous services have two port types. Each port type performs a one-way operation: one port type initiates the asynchronous process and the other calls back the client with the asynchronous response. In this example shown in [Example 14-1](#), the `portType` `LoanService` receives the client's loan application request and the `portType` `LoanServiceCallback` asynchronously calls back the client with the loan offer response.

Example 14–1 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>

```

14.2.2.2 partnerLinkType Section of the LoanService.wsdl File

The `partnerLinkType` section of the `LoanService.wsdl` file defines the following characteristics of the conversation between the BPEL process service component and the loan application approver Web service:

- The role (operation) played by each
- The `portType` provided by each for receiving messages within the context of 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 14–2](#), 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 14–2 partnerLinkType Definition

```

<!-- the LoanService partnerLinkType binds the service and
requestor portType into an asynchronous conversation.
-->
<plnk:partnerLinkType name="LoanService">
  <plnk:role name="LoanServiceProvider">
    <plnk:portType name="tns:LoanService"/>
  </plnk:role>
  <plnk:role name="LoanServiceRequester">
    <plnk:portType name="tns: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.

14.2.2.3 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 of the `LoanBroker.bpel` file. 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 14–3](#) provides an example.

Example 14–3 *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.

14.2.2.4 Composite Application File

Open the `composite.xml` file of `samples\tutorials\105.AsyncCompositeLoanBroker`. The loan application approver Web service appears, as shown in [Example 14–4](#).

Example 14–4 *Loan Application Approver Web Service*

```
<component name="LoanBroker">
  <implementation.bpel process="LoanBroker.bpel"/>
</component>
```

For more information, see [Section 14.2.1.1, "Step 1: Adding a Partner Link for an Asynchronous Service"](#) for instructions on creating a partner link.

14.2.2.5 Invoke and Receive Activities

View the `variables` and `sequence` sections of the `LoanBroker.bpel` file. Two areas of particular interest concern the `invoke` and `receive` activities:

- An `invoke` activity invokes a synchronous Web service (as discussed in [Chapter 13, "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 14-5](#) provides an example.

Example 14-5 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 WS-Addressing (described in [Section 14.3, "Using WS-Addressing in an Asynchronous Service"](#)). Because multiple processes may be waiting for service callbacks, Oracle BPEL 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 Oracle BPEL Server to correlate the response with the appropriate requesting instance.

For more information, see the following sections for instructions on creating `invoke` and `receive` activities:

- [Section 14.2.1.2, "Step 2: Adding an Invoke Activity"](#)
- [Section 14.2.1.3, "Step 3: Adding a Receive Activity"](#)

14.2.2.6 createInstance Attribute for Starting a New Instance

You may have noticed a `createInstance` attribute in the initial `receive` activity of the `sequence` section of the `LoanBroker.bpel` file. 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 14-6](#) shows the source code for the `createInstance` attribute:

Example 14–6 createInstance Attribute

```
<!-- receive input from requestor -->
<receive name="receiveInput" partnerLink="client"
  portType="tns:LoanBroker"
  operation="initiate" variable="input"
  createInstance="yes"/>
```

14.2.2.7 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. [Figure 14–1](#) shows an example of a dehydration point in the loan application approver Web service.

14.3 Using WS-Addressing in an Asynchronous Service

Because there can be many active instances at any given point in time, Oracle BPEL 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 14–5](#) provides an overview of Web Services Addressing (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 14–5 Callback with WS-Addressing Headers

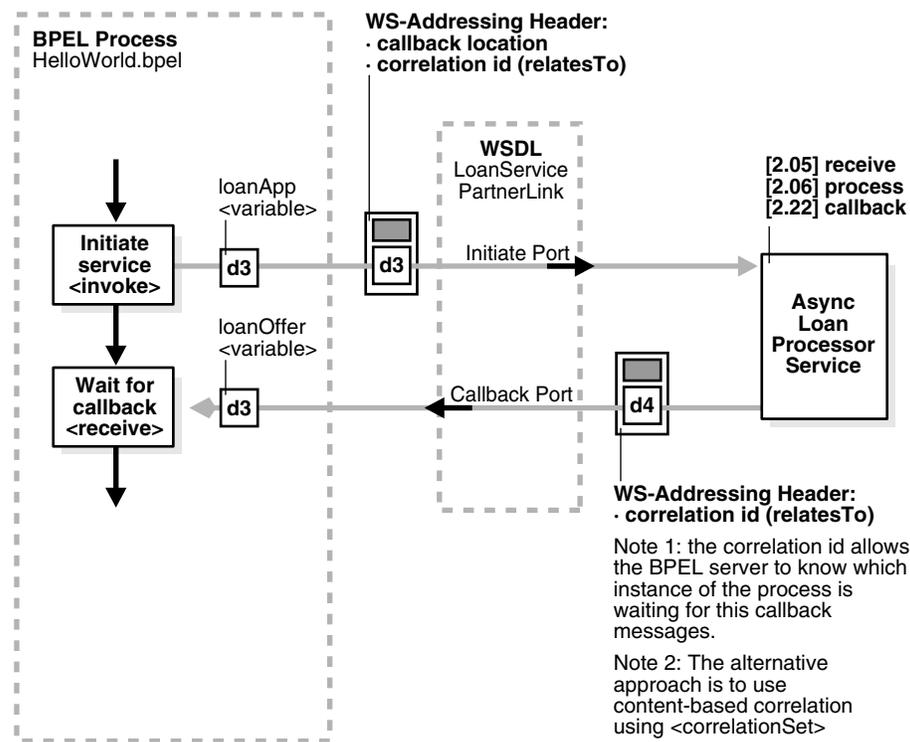


Figure 14–5 shows how messages are passed along with WS headers so that the response can be sent to the correct destination.

The example in this chapter uses WS-Addressing for correlation. To view the messages, you can use TCP tunneling, which is described in [Section 14.3.1.1, "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.

14.3.1 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.

14.3.1.1 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 with Web services and BPEL process service components 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, as well as displaying 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 Oracle BPEL 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.

14.3.1.1.1 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. Start your TCP listener to listen on a port such as 1234 and send on a port such as 9700 (port 9700 is used in this example and is the default after Oracle BPEL Process Manager for Developers installation). If you installed Oracle BPEL Process Manager as part of an Oracle Application Server SOA install type, substitute the correct port number throughout these instructions. For example, you can use the TCP tunnel included with Apache Axis (bundled with Oracle BPEL Process Manager) by executing the following from the operating system command prompt:

```
prompt> obsetenv  
prompt> java -classpath %OB_CLASSPATH% orabpel.apache.axis.utils.tcpmon 1234  
localhost 9700
```

2. Add an `endpointURI` property in the `componentType` file for your flow to override the endpoint of the service. For example, to see the messages exchanged between the LoanFlow demo sample and the CreditRatingService that it calls, change the definition of the CreditRatingService location in the `componentType` file found in `SOA_Oracle_Home\bpel\samples\demos\LoanDemo\LoanFlow`.

3. Compile and deploy the LoanDemo from the operating system command prompt:

```
prompt> cd SOA_Oracle_Home\bpel\samples\demos\LoanDemo  
prompt> ant
```

Note that while the CreditRatingService is also a BPEL process service component, 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.

For more information, download the TCP Monitor tool from <http://ws.apache.org/commons/tcpmon/>.

14.3.1.1.2 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:

Note: The `optSoapShortcut` and `SoapServerURL` parameters are not enabled for beta 3.

1. Start a TCP listener to listen on a port such as 9710 and to send on the Oracle BPEL Process Manager port (for example, 9700 is the default after installation of Oracle BPEL Process Manager for Developers).
2. Turn off the optimization of local SOAP calls performed by Oracle BPEL Process Manager to see the impact of changing the callback port:
 - a. Go to Oracle Enterprise Manager 10g Application Server Control Console.
 - b. Go to the **Configuration** tab for the deployed composite application.
 - c. Scroll down to the `optSoapShortcut` property.
 - d. Change the value from **true** to **false**.
3. Go to the `SoapServerUrl` property on the **Configuration** tab.
4. Change this property to `http://localhost:9710`.
5. Click the **Apply** button.
6. Restart Oracle BPEL Server to initialize these changes and initiate any flow that invokes asynchronous Web services (for example the LoanFlow demonstration). You can combine this with the synchronous TCP tunneling configuration to send the UnitedLoan service initiation request through your first TCP tunnel.

The callbacks from the asynchronous services are shown in the TCP listener, such as the UnitedLoan service callback.

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.

For more information, see the following documents:

- *Web Services Addressing (WS-Addressing) Specification* for complete details about WS-Addressing, which is accessible from <http://www.oracle.com/technology/bpel>
- `SOA_Oracle_Home/bpel/samples/demos/LoanDemo` for the LoanFlow demo used in this section

14.4 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 more information, see the following correlation set example at *SOA_Oracle_Home\bpel\samples\tutorials\109.CorrelationSets*.

14.4.1 How to Use Correlation Sets in an Asynchronous Service

This section contains the following topics:

- [Section 14.4.1.1, "Step 1: Creating a Project"](#)
- [Section 14.4.1.2, "Step 2: Configuring Partner Links and File Adapter Services"](#)
- [Section 14.4.1.3, "Step 3: Creating Three Receive Activities"](#)
- [Section 14.4.1.4, "Step 4: Creating Correlation Sets"](#)
- [Section 14.4.1.5, "Step 5: Associating Correlation Sets with Receive Activities"](#)
- [Section 14.4.1.6, "Step 6: Creating Property Aliases"](#)
- [Section 14.4.1.7, "Step 7: Reviewing WSDL File Content"](#)

14.4.1.1 Step 1: Creating a Project

1. Start Oracle JDeveloper.
2. Select **New > Application** from the **File** main menu.
3. Enter the following values:

Field	Value
Application Name	Enter an application name (for this example, MyCorrelationSetApp).
Application Template	Select No Template [All Technologies] .

4. Accept the default values for all remaining settings, and click **OK**.
5. Click **Cancel** in the Create Project window.
6. Select the **Application Menu** list for the application you just created.
7. Select **New Project** from the list that appears.
The New Gallery window appears.
8. Double-click **SCA Project** in the **Items** field.
The Create SCA Project window appears.
9. Enter the following values:

Field	Value
Project Name	Enter a name in the field (for this example, MyCorrelationSetComposite).

Field	Value
Upon project creation, create	Click the check box and select BPEL Process from the list. This creates an SCA project with a BPEL process.

10. Click **OK**.

The Create BPEL Process window appears.

11. Enter the following values:

Field	Value
Name	Enter MyCorrelationSet .
Namespace	Accept the default value.
Template	Select Asynchronous BPEL Process .
Expose as Composite Service	Select the check box. After process creation, note the SOAP adapter service that appears in the Services swimlane. This service provides the entry point to the composite application from the outside world.

12. Click **Finish**.

14.4.1.2 Step 2: Configuring Partner Links and File Adapter Services

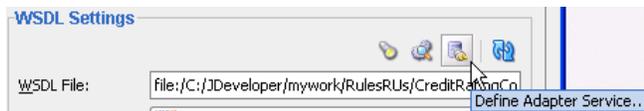
You now create three partner links that use the adapter services.

This section contains these topics:

- [Section 14.4.1.2.1, "Creating an Initial Partner Link and File Adapter Service"](#)
You create an initial partner link with an adapter service for reading a loan application.
- [Section 14.4.1.2.2, "Creating a Second Partner Link and File Adapter Service"](#)
You create a second partner link with an adapter service for reading an application response.
- [Section 14.4.1.2.3, "Creating a Third Partner Link and File Adapter Service"](#)
You create a third partner link with an adapter service for reading a customer response.

14.4.1.2.1 Creating an Initial Partner Link and File Adapter Service

1. Double-click the **MyCorrelationSet** BPEL process.
2. Expand **BPEL Services** in the **Component Palette**.
3. Drag and drop an initial **PartnerLink** activity into the right swim lane of the designer window.
4. Click the third icon at the top (the **Define Adapter Service** icon). This starts the Adapter Configuration Wizard, as shown in [Figure 14-6](#).

Figure 14–6 Adapter Configuration Wizard Startup

5. Select **File Adapter** on the Configure Service or Adapter window and click **Next**.
6. Click **Next** on the Welcome window.
7. Enter **FirstReceive** in the **Service Name** field on the Service Name window and click **Next**.
8. Select **Read File** as the **Operation Type** on the Operation window and click **Next**. The **Operation Name** field is automatically filled in with **Read**.
9. Select **Directory Names are Specified as Physical Path**.
10. Click **Browse** above to the **Directory for Incoming Files (physical path)** field.
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. Enter appropriate file filtering parameters in the File Filtering window.
15. Click **Next**.
16. Enter appropriate file polling parameters in the File Polling window.
17. Click **Next**.
18. Click **Browse** next to the **Schema Location** field in the Messages window to display the Type Chooser window.
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 window. All other fields are automatically completed. The window looks as follows:

Field	Value
Name	FirstReceive
WSDL File	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**.

14.4.1.2.2 Creating a Second Partner Link and File Adapter Service 1. Drag and drop a second **PartnerLink** activity below the **FirstReceivePL partner link** activity.

2. Click the third icon at the top (the **Define Adapter Service** icon).
3. Click **Next** on the Welcome window.
4. Select **File Adapter** in the Adapter Type window and click **Next**.
5. Enter **SecondFileRead** in the **Service Name** field on the Service Name window and click **Next**. This name must be unique from the one you entered in Step 7 on page 14-16.
6. Select **Read File** as the **Operation Type** in the Operation window
7. Change the name in the **Operation Name** field to **Read1**.
8. Click **Next**.
9. Select **Directory Names are Specified as Physical Path**.
10. Click **Browse** above to the **Directory for Incoming Files (physical path)** field.
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 window.
15. Click **Next**.
16. Enter appropriate file polling parameters in the File Polling window.
17. Click **Next**.
18. Click **Browse** next to the **Schema Location** field in the Messages window to display the Type Chooser window.
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 window. All other fields are automatically completed. The window looks as follows:

Field	Value
Name	SecondReceive
WSDL File	file:/C:/JDeveloper/mywork/Application_Name/SOA_Project_Name/SecondFileRead.wsdl where C:/JDeveloper represents the Oracle JDeveloper home directory for this example.
Partner Link Type	Read1_plt

Field	Value
Partner Role	Leave unspecified.
My Role	Read1_role

23. Click **OK**.

14.4.1.2.3 Creating a Third Partner Link and File Adapter Service

1. Drag and drop a third **PartnerLink** activity below the **SecondReceivePL partner link** activity.
2. Click the third icon at the top (the **Define Adapter Service** icon).
3. Click **Next** on the Welcome window.
4. Select **File Adapter** in the Adapter Type window and click **Next**.
5. Enter **ThirdFileRead** in the **Service Name** field of the Service Name window and click **Next**. This name must be unique from the one you entered in Step 7 on page 14-16 and Step 5 on page 14-17.
6. Select **Read File** as the **Operation Type** in the Operation window
7. Change the name in the **Operation Name** field to **Read2**. This name must be unique.
8. Click **Next**.
9. Select **Directory Names are Specified as Physical Path**.
10. Click **Browse** above to the **Directory for Incoming Files (physical path)** field.
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 window.
15. Click **Next**.
16. Enter appropriate file polling parameters in the File Polling window.
17. Click **Next**.
18. Click **Browse** next to the **Schema Location** field in the Messages window to display the Type Chooser window.
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 window. All other fields are automatically completed. The window looks as follows:

Field	Value
Name	ThirdReceive
WSDL File	file:/C:/JDeveloper/mywork/Application_Name/SOA_Project_Name/ThirdFileRead.wsdl where C:/JDeveloper 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 window looks as shown in [Figure 14-7](#):

Figure 14-7 BPEL Process Design



14.4.1.3 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.

14.4.1.3.1 Creating an Initial Receive Activity

1. Expand **BPEL Activities** in the **Component Palette**.
2. Drag and drop a **Receive** activity from the **Process Activities** list of the **Component Palette** section to below the **receiveInput** receive activity in the designer window.
3. Double-click the **receive** icon to display the Receive window.
4. Enter the following details to associate the first partner link (**FirstReceive**) with the first receive activity:

Field	Value
Name	receiveFirst
Partner Link	FirstReceive
Create Instance	Select this check box.

The **Operation (Read)** field is automatically filled in.

5. Click the first icon to the right of the **Variable** field. This is the automatic variable creation icon.
6. Click **OK** on the Create Variable window that appears.
A variable named **receiveFirst_Read_InputVariable** is automatically created in the **Variable** field.
7. Ensure that you selected the **Create Instance** check box, as mentioned in Step 4.
8. Click **OK**.

- 14.4.1.3.2 Creating an Initial Receive Activity**
1. Drag and drop a second **Receive** activity from the **Component Palette** section to below the **receiveFirst receive** activity.
 2. Double-click the **receive** icon to display the Receive window.
 3. Enter the following details to associate the second partner link (**SecondReceivePL**) with the second receive activity:

Field	Value
Name	receiveSecond
Partner Link	SecondFileRead
Create Instance	Do <i>not</i> select this check box.

The **Operation (Read1)** field is automatically filled in.

4. Click the first icon to the right of the **Variable** field.
5. Click **OK** on the Create Variable window that appears.
A variable named **receiveSecond_Read1_InputVariable** is automatically created in the **Variable** field.
6. Click **OK**.

14.4.1.3.3 Creating a Third Receive Activity

1. Drag and drop a third **Receive** activity from the **Component Palette** section to below the **receiveSecond receive** activity.
2. Double-click the **receive** icon to display the Receive window.
3. Enter the following details to associate the third partner link (**ThirdReceivePL**) with the third receive activity:

Field	Value
Name	receiveThird
Partner Link	ThirdFileRead
Create Instance	Do <i>not</i> select this check box.

The **Operation (Read2)** field is automatically filled in.

4. Click the first icon to the right of the **Variable** field.
5. Click **OK** on the Create Variable window that appears.

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.

14.4.1.4 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.

14.4.1.4.1 Creating an Initial Correlation Set

1. Right-click **Correlation Sets** and select **Expand All Child Nodes** in the **Structure** window of Oracle JDeveloper.
2. Right-click the second **Correlation Sets** folder and select **Create Correlation Set**.
3. Enter **CorrelationSet1** in the **Name** field of the Create Correlation Set window.
4. Click the + sign in the **Properties** section to display the Property Chooser window.
5. Select **Properties**, then click the + sign (first icon at the top) to display the Create Correlation Set Property window.
6. Enter **NameCorr** in the **Name** field and click the **flashlight** icon to the right of the **Type** field.
7. Select **string** in the Type Chooser window and click **OK**.
8. Click **OK** to close the Create Correlation Set Property window, the Property Chooser window, and the Create Correlation Set window.

14.4.1.4.2 Creating 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. Enter **CorrelationSet2** in the **Name** field of the Create Correlation Set window.
4. Click the + sign in the **Properties** section to display the Property Chooser window.
5. Select **Properties**, then click the + sign to display the Create Correlation Set Property window.
6. Enter **IDCorr** in the **Name** field and click the **flashlight** icon to the right of the **Type** field.
7. Select **double** in the Type Chooser window and click **OK**.
8. Click **OK** to close the Create Correlation Set Property window, the Property Chooser window, and the Create Correlation Set window.

14.4.1.5 Step 5: Associating Correlation Sets with Receive Activities

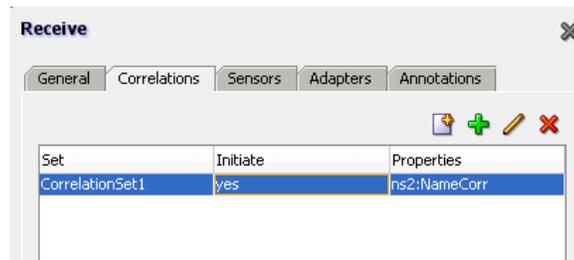
You now associate the correlation sets with the receive activities. You perform the following correlation set tasks:

- 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.

14.4.1.5.1 Associating the First Correlation Set with a Receive Activity

1. Double-click the **receiveFirst receive** activity to display the Receive window.
2. Click the **Correlations** tab.
3. Click the second + sign to display the Correlation Set Chooser window.
4. Select **CorrelationSet1**, then click **OK**.
5. Set the **Initiate** column to **yes**, as shown in [Figure 14–8](#). When set to **yes**, the set is initiated with the values of the properties occurring in the message being exchanged.

Figure 14–8 Initiate Column Set to yes in Receive Dialog

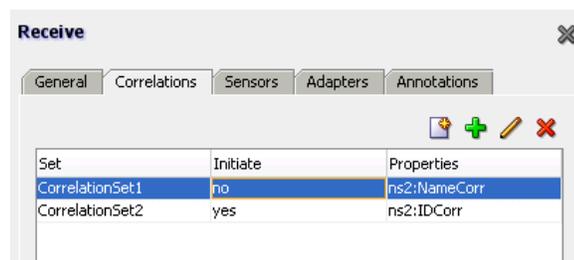


6. Click **OK**.

14.4.1.5.2 Associating the Second Correlation Set with a Receive Activity

1. Double-click the **receiveSecond receive** activity to display the Receive window.
2. Click the **Correlations** tab.
3. Click the second + sign to display the Correlation Set Chooser window.
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**, as shown in [Figure 14–9](#).

Figure 14–9 Initiate Column Set to no in Receive Dialog



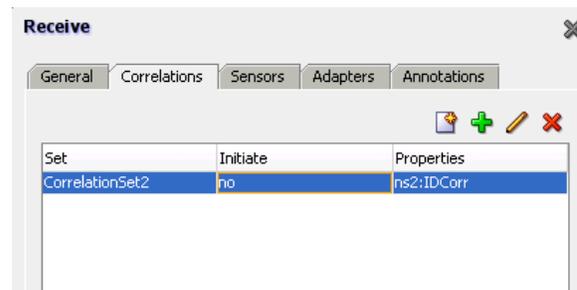
9. Click **OK**.

This groups the first and second receive activities into a correlated group.

14.4.1.5.3 Associating the Third Correlation Set with a Receive Activity

1. Double-click the **receiveThird receive** activity to display the Receive window.
2. Click the **Correlations** tab.
3. Click the second + sign to display the Correlation Set Chooser window.
4. Select **CorrelationSet2**, then click **OK**.
5. Set the **Initiate** column to **no** for **CorrelationSet2**, as shown in [Figure 14–10](#).

Figure 14–10 *Initiate Column Set to no in Receive Dialog*



6. Click **OK**.

This groups the second and third receive activities into a second correlated group.

14.4.1.6 Step 6: Creating Property Aliases

Property aliases enable you to map a global property to a field in a specific message part. This enables the property name to become an alias for the message part and location. The alias can be used in XPath expressions.

14.4.1.6.1 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).
1. Right-click **Property Aliases** in the **Structure** section of Oracle JDeveloper.
 2. Select **Create Property Alias**.
 3. Select **NameCorr** in the **Property** list.
 4. Expand and select **Message Types > Web Services > FirstReceivePL > FirstReceive.wsdl > Message Types > LoanAppl_msg > Part - LoanAppl**.
 5. Press **Ctrl** and then the **space bar** in the **Query** field 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. Press **Ctrl** and then the **space bar** in the **Query** field to define the following XPath expression:

```
/ns4:LoanAppResponse/ns4:APR
```

14.4.1.6.2 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).
1. Right-click **Property Aliases** in the **Structure** section.
 2. Select **Create Property Alias**.
 3. Select **IDCorr** in the **Property** list.
 4. Expand and select **Message Types > Project WSDL Files > SecondFileRead.wsdl > Message Types > LoanAppResponse_msg > Part - LoanAppResponse**.
 5. Press **Ctrl** and then the **space bar** in the **Query** field 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. Press **Ctrl** and then the **space bar** in the **Query** field to define the following XPath expression:

```
/ns6:CustResponse/ns6:APR
```

Design is now complete.

14.4.1.7 Step 7: Reviewing 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 14–7](#) provides an example.

Example 14–7 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 14-8](#):

Example 14-8 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 14-9](#) provides an example.

Example 14-9 WSDL File Import

```
<import namespace="http://xmlns.oracle.com/MyCorrelationSet"
location="MyCorrelationSet.wsdl" />
```

14.5 Use Case for Invoking a Asynchronous Web Service

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 desired information.

Note: The sample demonstrated in `BPELCallingAsyncAXIS` is not included in the `soa-samples.zip` file for beta 3.

Parallel Flow in a BPEL Process

Parallel flows enable a BPEL process service component to perform multiple tasks at the same time, which is especially useful when you need to perform several time-consuming and independent tasks.

This chapter includes the following sections:

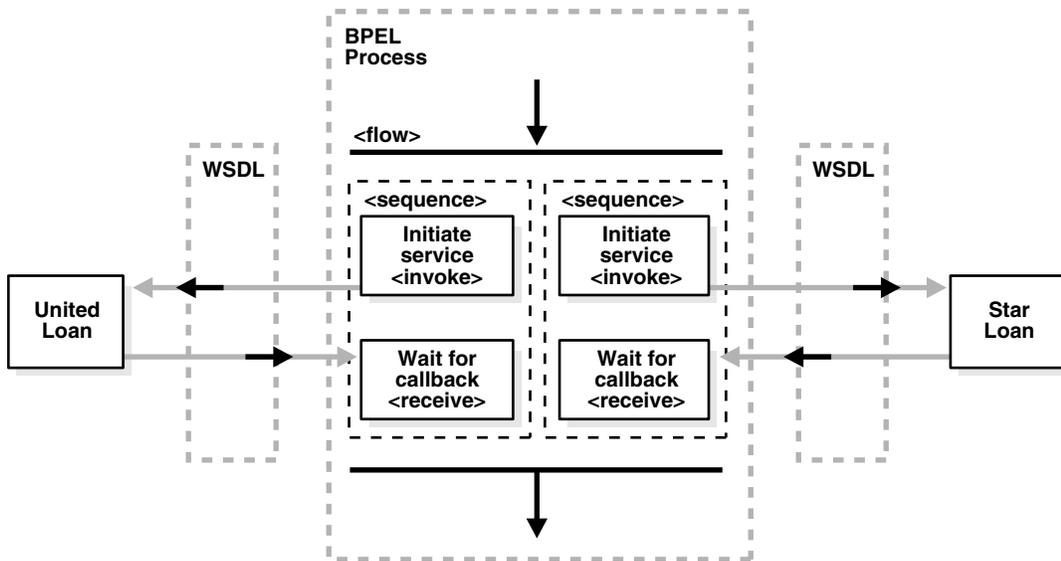
- [Section 15.1, "Introduction to Parallel Flows in BPEL Processes"](#)
- [Section 15.2, "Creating a Parallel Flow in a BPEL Process"](#)
- [Section 15.3, "Customizing the Number of Flow Activities with the flowN Activity"](#)
- [Section 15.4, "Use Case for Parallel Flows"](#)

15.1 Introduction to Parallel Flows in BPEL Processes

Sometimes a BPEL process service component must 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 once, and receive the responses as they come in. This method is much more time efficient.

[Figure 15-1](#) provides an overview of a BPEL process service component performing a parallel flow to retrieve loan offers from two different Web services. Here, two asynchronous callbacks execute in parallel, so that one callback does not have to wait for the other to complete first. Each response is stored in a different global variable.

Figure 15-1 Parallel Flow Invocation



15.2 Creating a Parallel Flow in a BPEL Process

15.2.1 How to Create a Parallel Flow in a BPEL Process

1. From the Component Palette, drag a flow activity into the designer.

15.3 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 run time 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 15–2 shows an Oracle Enterprise Manager 11g Application Server Control Console view of a **flowN** activity 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 15–2 Oracle Enterprise Manager 11g Application Server Control Console View of the Execution of a flowN activity

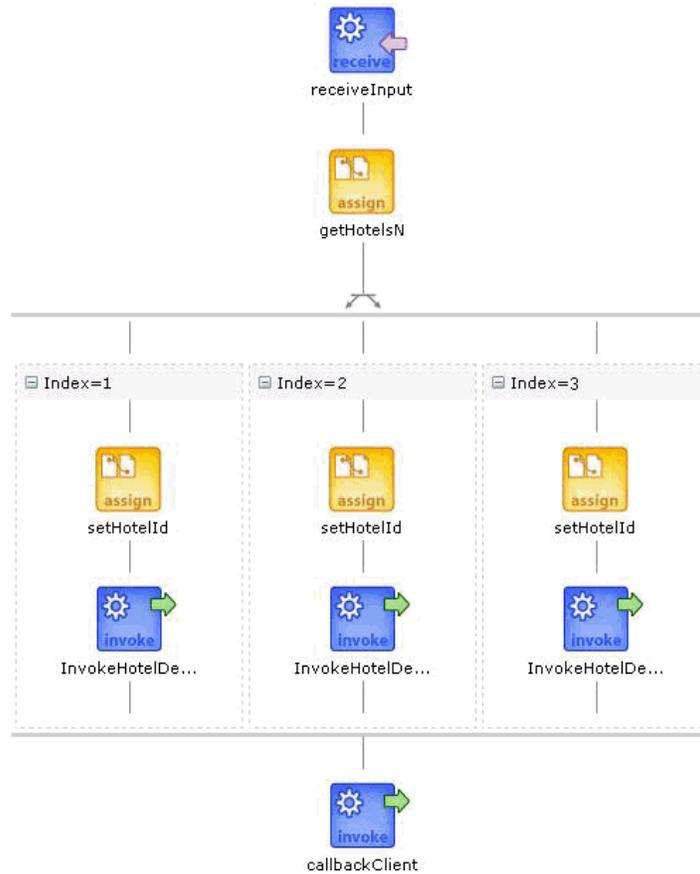


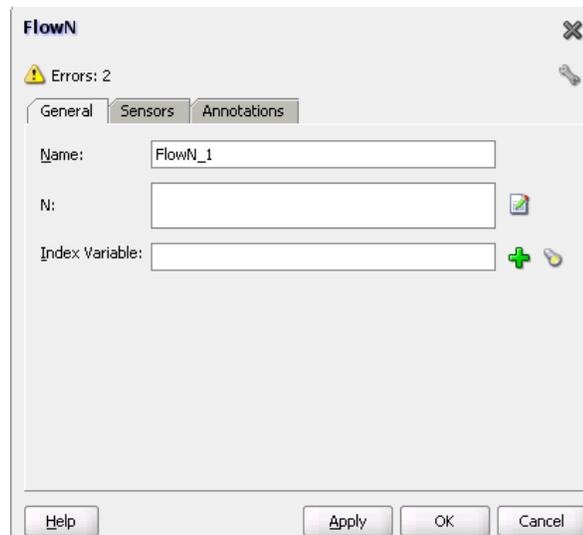
Figure 15–3 shows how a **flowN** activity appears in Oracle JDeveloper.

Figure 15–3 FlowN Activity Setup in the Diagram Window



Figure 15–4 shows the flowN Window, which appears when you double-click the flowN activity.

Figure 15–4 flowN Window



The flowN windows enables you to name the **flowN** activity, enter an expression for calculating the value of N, and define the index variable.

15.3.1 How to Create a flowN Activity

1. From the Component Palette, drag a flowN activity into the designer.

15.3.2 What Happens When You Create a FlowN Activity

The following code is a reference implementation from a .bpel file that uses the flowN activity to look up information on an arbitrary number of hotels. The following actions take place:

1. First, you name the sequence, as shown in [Example 15-1](#):

Example 15-1 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"
  -->
```

2. The receive activity calls the client partner link to get the information that the flowN activity needs to define N and look up hotel information. [Example 15-2](#) provides an example.

Example 15-2 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.
  For lisibility, an intermediate varaible 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
  -->
```

3. 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 15-3](#) provides an example.

Example 15-3 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.
    -->
```

- Next, the copy rule shown in [Example 15–4](#) uses the index variable to concatenate the hotel entries into a list:

Example 15–4 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 15–5](#) provides an example.

Example 15–5 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 procees doesn't do anything with the retrieved inforamtion.
In the real life, it could be then used to continue the process.
Note: Meanwhile an indexing variable is used, unlike a while loop, the
activities a executed in parallel, not sequentially.
-->
  </sequence>
</bpelx:flowN>
```

- Finally, the BPEL process sends detailed information on each hotel to the client partner link. [Example 15–6](#) provides an example.

Example 15–6 Invoke Activity

```
<invoke name="callbackClient" partnerLink="client"
portType="client:NflowHotelsCallback" operation="onResult"
inputVariable="outputVariable"/>
</sequence>
</sequence>
```

15.4 Use Case for Parallel Flows

In [Chapter 14, "Invoking an Asynchronous Web Service from a BPEL Process"](#) you learned how to call an asynchronous Web service for the United Loan service. Because the United Loan service can take up to several days to return a loan offer but you need to collect another loan offer from Star Loan, you can define your BPEL process service component so both tasks run in parallel.

This use case shows how to program the BPEL flow to perform two asynchronous callbacks to loan services in parallel.

Note: The sample demonstrated in `106.ParallelFlows` is not included in the `soa-samples.zip` file for beta 3.

Conditional Branching in BPEL Processes

This chapter describes conditional branching. Conditional branching introduces decision points to control the flow of execution of a BPEL process service component.

This chapter includes the following sections:

- [Section 16.1, "Introduction to Conditional Branching"](#)
- [Section 16.2, "Adding a Switch Activity to Define Conditional Branching"](#)
- [Section 16.3, "Adding a While Activity to Define Conditional Branching"](#)
- [Section 16.4, "Use Case for Conditional Branching"](#)

16.1 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:** In this method, you set up two or more branches, with each branch in the form of an XPath expression. If the expression is true, then the branch is executed. If the expression is false, then the BPEL process 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 more than one branch condition is true, then BPEL executes the first true branch. [Section 16.2, "Adding a Switch Activity to Define Conditional Branching"](#) on page 16-1 explains how to create switch activities.
- **While activity:** You can use a while activity to create a while loop to select between two actions. [Section 16.3, "Adding a While Activity to Define Conditional Branching"](#) on page 16-4 describes while activities.

A number of 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 of time, the BPEL flow can stop waiting and resume its activities. [Chapter 19, "Events and Timeouts in BPEL Processes"](#) explains this feature in detail.

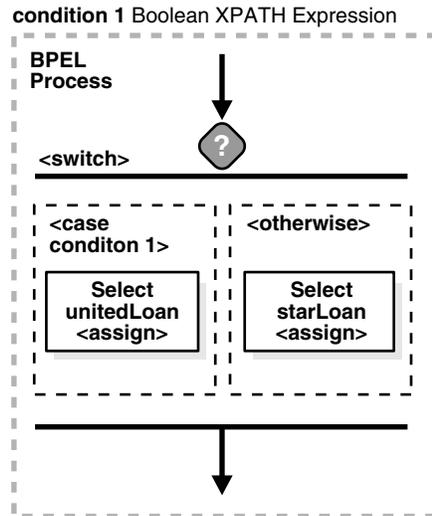
16.2 Adding a Switch Activity to Define Conditional Branching

In [Chapter 15](#), the flow activity of the BPEL process service component gathered two loan offers 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 offers and make decisions based on that comparison, the BPEL flow requires a switch activity.

Figure 16–1 provides an overview of a BPEL conditional branching process that has been defined in a switch activity.

Figure 16–1 Conditional Branching



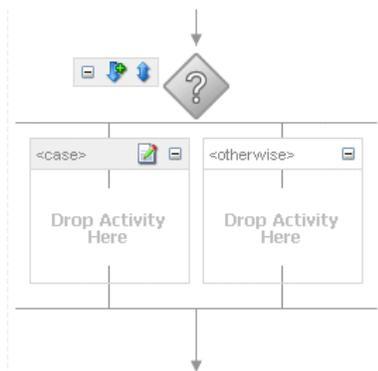
16.2.1 How to Add a Switch Activity

To add a switch activity to your BPEL flow in Oracle JDeveloper:

1. Select **BPEL** from the **Component Palette** list.
2. Expand BPEL Activities.
3. Drag a **switch** activity from the **BPEL Activities** list into your BPEL flow.
4. Click the + sign to expand the **switch** activity, as shown in Figure 16–2.

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 16–2 Switch Activity



5. Right-click the first branch and select **Edit** from the menu.

The Switch Case window appears.

6. Enter an XPath Boolean expression in the **Expression** field by pressing the **Ctrl** key and then the **space** bar to start the XPath Building Assistant. [Example 16–1](#) provides an example:

Example 16–1 XPath Expression

```
bpws:getVariableData('loanOffer1','payload','/loanOffer/APR') >
bpws:getVariableData('loanOffer2','payload','/loanOffer/APR')
```

7. Enter this expression on one line. To use the XPath Expression Builder, click the **XPath Expression Builder** icon above the **Expression** field.

The two loan offers that the LoanFlow tutorial uses 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 `loanOffer2`'s payload to `selectedLoanOffer`'s payload.
- If `loanOffer1` does *not* have the lower APR than `loanOffer2`, then the otherwise case assigns `loanOffer1`'s payload to `selectedLoanOffer`'s payload.

For more information, see the [SOA_Oracle_Home\bpel\samples\references\Switch](#) example of creating switch activities in Oracle JDeveloper.

16.2.2 What Happens When You Add a Switch Activity

A switch activity, like a flow activity, has multiple branches. In [Example 16–2](#), there are only two branches. The first branch, which selects a loan offer from United Loan, is executed if a case condition containing an XPath Boolean expression is met. Otherwise, the second branch, which selects the Star Loan loan offer, is executed. By default, the switch activity provides two switch cases, but you can add more if you want.

Example 16–2 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>

```

16.3 Adding 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.

16.3.1 How To Add a While Activity

To create a while activity in Oracle JDeveloper:

1. Drag and drop a **while** activity from the **BPEL Activities** list of the **Component Palette** into your BPEL flow.

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 drop an activity to define the while loop. [Figure 16–3](#) provides an example.

Figure 16–3 While Activity



2. Drag the activity that you want to use to define the while condition onto the **Drop Activity Here** area of the **while** activity.

The activity can be an existing activity or a new activity, such as an **invoke** activity to launch a task.

For more information, see *SOA_Oracle_Home\bpel\samples\references\While* for examples of defining a while activity in Oracle JDeveloper.

16.4 Use Case for Conditional Branching

The BPEL process service component you created in [Chapter 15, "Parallel Flow in a BPEL Process"](#) collected two loan offers, one from United Loan and another from Star Loan. This chapter describes how to design the BPEL process service component to select the loan with the lowest annual percentage rate (APR) automatically.

Note: The sample demonstrated in LoanFlow is not included in the soa-samples.zip file for beta 3.

Fault Handling in BPEL Processes

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 run-time faults.

This chapter contains the following topics:

- [Section 17.1, "Use Case for Fault Handling"](#)
- [Section 17.2, "Defining a Fault Handler"](#)
- [Section 17.3, "BPEL Standard Faults"](#)
- [Section 17.4, "Categories of BPEL Faults"](#)
- [Section 17.5, "Getting Fault Details with the `getFaultAsString` XPath Extension Function"](#)
- [Section 17.6, "Using the Scope Activity to Manage a Group of Activities"](#)
- [Section 17.7, "Throwing Internal Faults"](#)
- [Section 17.8, "Returning External Faults"](#)
- [Section 17.9, "Using a Fault Handler within a Scope"](#)
- [Section 17.10, "Using Compensation After Undoing a Series of Operations"](#)
- [Section 17.11, "Using the Terminate Activity to Stop a Business Process Instance"](#)

17.1 Use Case for Fault Handling

This chapter uses an example of a credit rating service returning a negative credit message instead of a credit rating number. You also learn how to add a fault handler to a BPEL process service component to handle the message.

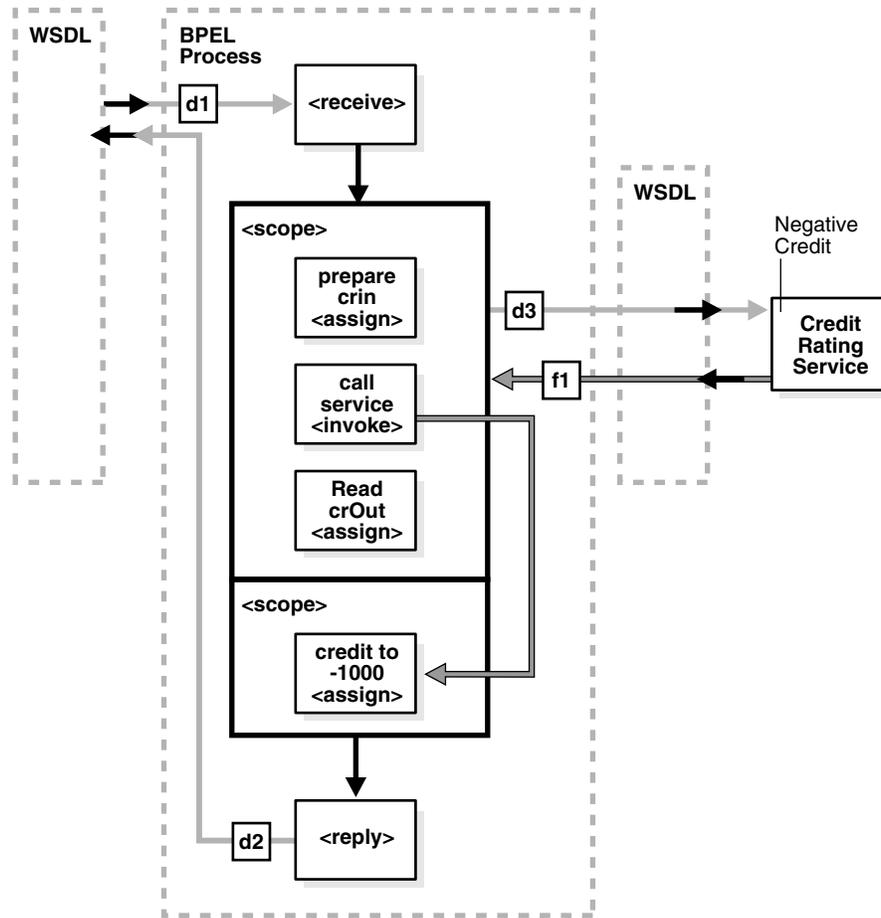
Note: The sample demonstrated with `ResilientDemo` is not included in the `soa-samples.zip` file for beta 3.

17.2 Defining 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 17-1](#) shows how a fault handler sets the credit rating variable at `-1000`.

Figure 17-1 Fault Handling



The following code segment defines the fault handler for this operation:

```

<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.

17.3 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/`:

- `selectionFailure`
- `conflictingReceive`
- `conflictingRequest`
- `mismatchedAssignmentFailure`
- `joinFailure`
- `forcedTermination`
- `correlationViolation`
- `uninitializedVariable`
- `repeatedCompensation`
- `invalidReply`

Standard faults are defined as follows:

- Typeless, meaning they do not have associated `messageTypes`
- Not associated with any WSDL message
- Caught without a fault variable:

```
<catch faultName="bpws:selectionFault">
```

17.4 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
- Run-time faults

17.4.1 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">
```

17.4.2 Run-time Faults

Run-time 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 SOAP fault occurs in a SOAP call, an exception is thrown by Oracle BPEL Server, and so on.

Oracle BPEL Server includes several run-time faults. These faults are included in the `http://schemas.oracle.com/bpel/extension` namespace. These faults are associated with the message type `RuntimeFaultMessage`. The following WSDL file defines the message type:

```
<?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 fault variable (of message type `RuntimeFaultMessage`) is used when catching the fault, the fault code can be queried from the fault variable, along with the fault summary and detail.

17.4.2.1 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. [Table 17-1](#) describes the fault codes.

Table 17-1 *bindingFault Fault Codes*

Fault Code	Description of Fault
<code>VersionMismatch</code>	The processing party found an invalid namespace for the SOAP envelope element.
<code>MustUnderstand</code>	An immediate child element of the SOAP header element that was either not understood or not obeyed by the processing party contained a SOAP <code>MustUnderstand</code> attribute with a value of 1.
<code>Client.GenericError</code>	Generic error on the client side
<code>Client.WrongNumberOfInputParts</code>	Input message part number mismatch
<code>Client.WrongNumberOfOutputParts</code>	Output message part number mismatch
<code>Client.WrongTypeOfInputPart</code>	Input message part type error
<code>Client.WrongTypeOfOutputPart</code>	Output message part type error
<code>Server.GenericError</code>	Generic error on the server side
<code>Server.NoService</code>	Server is up, but there is no service
<code>Server.NoHTTPSOAPAction</code>	Request is missing the HTTP SOAP action
<code>Server.Unauthenticated</code>	Request is not authenticated
<code>Server.Unauthorized</code>	Request is not authorized

17.4.2.2 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. A `remoteFault` can be configured to be retried. [Table 17–2](#) describes the fault codes.

Table 17–2 *remoteFault* Fault Codes

Fault Code	Description of Fault
ConnectionRefused	Remote server is unavailable
WSDLReadingError	Failed to read the WSDL
GenericRemoteFault	Generic remote fault

17.4.2.3 replayFault

A `replayFault` replays the activity inside a scope. At any point inside a scope, this fault is migrated up to the scope. Oracle BPEL Server then re-executes the scope from the beginning.

17.4.2.4 Catching Run-time Faults Example

BPEL run-time faults can be caught as a named BPEL fault. The `bindingFault` and `remoteFault` can be associated with a message. This enables the `faultHandler` to get details about the 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.

1. Import `RuntimeFault.wsdl` into your process WSDL (located under the `SOA_Oracle_Home\bpel\system\xml\lib` directory).
2. Declare a variable with `messageType bpel:RuntimeFaultMessage`.
3. Catch it using

```
<catch faultName="bpel:remoteFault" | "bpel:bindingFault"
faultName="varName">
```

See Also: The following sample, which describes how to handle run-time binding faults:

- `SOA_Oracle_Home\bpel\samples\demos\ResilientDemo`

17.5 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.

```
<catchAll>
  <sequence>
    <assign>
      <from expression="bpel:getFaultAsString()"/>
      <to variable="faultVar" part="message"/>
    </assign>
    <reply faultName="ns1:myFault" variable="faultVar" .../>
  </sequence>
```

```
</catchAll>
```

17.6 Using the Scope Activity to Manage a Group of Activities

The scope activity provides a container and a context for other activities. A scope provides handlers for faults, events, and compensation, as well as data variables and correlation sets. Using a scope activity simplifies a BPEL flow by grouping functional structures together. This allows you to collapse them into what appears to be a single element in Oracle JDeveloper.

The following code example shows a scope activity. In this case, the process for getting a credit rating based on a customer's social security number has been placed inside a scope named `getCreditRating`. This identifies functional blocks of code and sets them apart visually. In Oracle JDeveloper, you can collapse the activities contained inside the scope into a single visual element, or expand them when necessary.

```
<scope name="getCreditRating">
  <variables>
    <variable name="crError"
      messageType="services:CreditRatingServiceFaultMessage"/>
  </variables>
  <assign name="assign-2">
    <copy>
      <to variable="input" part="payload"
        query="/autoloan:loanApplication/autoloan:creditRating"/>
    </copy>
  </assign>
</sequence>
</scope>
```

To add a scope activity:

1. Click and drag a **scope** activity into the BPEL process service component diagram.
2. Open the **scope** by double-clicking it or by single-clicking the **+** sign.
3. Drag activities from the **Component Palette** to build the function within the scope.

17.7 Throwing Internal Faults

A BPEL application can generate and receive fault messages. The `throw` activity has three elements: its name, the name of the `faultName`, and the `faultVariable`. 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. Here is a code sample of a `throw` activity, which includes the fault elements, name, and partner link of the service to which the BPEL process service component sends the fault, and the copy rule that packages the message:

```
<throw name="delay" faultName="fault-1" faultVariable="fVar"/>
  <invoke name="invokeStockQuoteService" partnerLink="StockQuoteService"/>
  <assign>
    <copy>
      <from variable="response" part="result" query="/result"/>
      <to variable="output" part="payload" query="/tns:result"/>
    </copy>
  </assign>
```

See Also: The following documentation for examples of creating throw activities:

- [SOA_Oracle_Home\bpel\samples\references\Throw](#)

17.8 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.

17.8.1 Returning a Fault in a Synchronous Interaction

The syntax of a `reply` activity that returns a fault in a synchronous interaction is as follows:

```
<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, where 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.

See Also:

- [Chapter 16, "Conditional Branching in BPEL Processes"](#) for more information on setting up the conditional structure
- [Chapter 13, "Invoking a Synchronous Web Service from a BPEL Process"](#) for more information on synchronous interactions

17.8.2 Returning 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.

See Also:

- [Chapter 14, "Invoking an Asynchronous Web Service from a BPEL Process"](#) for more information on asynchronous interactions

17.9 Using 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. As described earlier, the scope activity includes 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. To catch any faults that are not already handled by name-specific catch activities, use the catchAll activity.

See Also: The following documentation for examples of creating fault handling:

- *SOA_Oracle_Home\bpel\samples\references\Catch*

17.9.1 Using the 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. The syntax to use an `empty` activity is as follows:

```
<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 17.11, "Using the Terminate Activity to Stop a Business Process Instance"](#) on page 17-9) had been performed.

Consider the following example:

```
<faulthandlers>
  <catch faultName="x:foo">
    <empty/>
  </catch>
  <catch faultVariable="bar">
    <empty/>
  </catch>
  <catch faultName="x:foo" faultVariable="bar">
    <empty/>
  </catch>
  <catchAll>
    <empty/>
  </catchAll>
</faulthandlers>
```

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.

17.10 Using Compensation After Undoing a Series of Operations

Compensation occurs when the BPEL process service component cannot complete a series of operations after some of them have already 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.

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 more than once are undefined.

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 as follows:

```
<compensate scope="ncname"? standard-attributes>
  standard-elements
</compensate>
```

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.

17.11 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.

The syntax for the terminate activity is as follows:

```
<terminate standard-attributes>  
  standard-elements  
</terminate>
```

See Also: The following documentation for examples of creating terminate activities:

- *SOA_Oracle_Home\bpel\samples\references\Terminate*

Incorporating Java and J2EE Code in BPEL Processes

You can incorporate sections of Java code and PL/SQL procedures into BPEL process service components in SOA applications.

This chapter includes the following sections:

- [Section 18.1, "Introduction to Java and J2EE Code in BPEL Concepts"](#)
- [Section 18.2, "Using Java Embedding in a BPEL Process"](#)
- [Section 18.3, "Using PL/SQL Procedures with SOA Applications"](#)

18.1 Introduction to Java and J2EE Code in BPEL Concepts

This chapter explains how to incorporate sections of Java code and PL/SQL procedures into a BPEL process. This is particularly useful when there is already Java code that can perform the desired function, and you want to use the existing code rather than start over with BPEL.

You can incorporate Java code using any of the following methods:

- [Section 18.1.1, "Use of Java Code Wrapped as a SOAP Service"](#)
- [Section 18.1.2, "Direct Embedding of Java Code in a BPEL Process"](#)
- [Section 18.1.3, "Using Java Code Wrapped in a Service Interface"](#)

18.1.1 Use of Java Code Wrapped as a SOAP Service

You can wrap the Java code as a 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.

However, a Java application wrapped as a SOAP service has the following drawbacks:

- It loses performance, because interactions are constantly being mapped back and forth between the Java code and the SOAP wrapper.
- It loses interoperability, 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).

18.1.2 Direct Embedding of Java Code in a BPEL Process

Another way to use Java in a BPEL process is to embed the code directly into the BPEL process using the Java BPEL `exec` extension `bpelx:exec`. The benefits of this approach are speed and transactionality. However, you can incorporate only fairly small segments of code. If you want to incorporate larger segments of code, or if the project requires the Java code to have the same look and feel throughout all the BPEL processes being created, consider wrapping it as a SOAP service.

18.1.2.1 Using the `bpelx:exec` Tag to Embed Java Code Snippets into a BPEL Process

The BPEL tag `bpelx:exec` enables you to embed a snippet of Java code within a BPEL process. 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 18-1](#) to embed the `invokeSessionBean` Java bean:

Example 18-1 `bpelx:exec` Tag

```
<bpelx:exec name="invokeSessionBean" language="java" version="1.4">
  <![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'"
          + ". Please make sure 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));
    } catch (NamingException ne) {
      addAuditTrailEntry(ne);
    } catch (ClassNotFoundException cnfe) {
      addAuditTrailEntry(cnfe);
    } catch (CreateException ce) {
      addAuditTrailEntry(ce);
    } catch (RemoteException re) {
      addAuditTrailEntry(re);
    }
  ]]>
</bpelx:exec>
```

```

]]>
</bpelx:exec>

```

18.1.2.2 Using 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 18–2](#) provides an example:

Example 18–2 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);
    ...

```

To generate the facade classes, use the `schemac` tool, which is provided with Oracle BPEL Process Manager. You can find the `schemac` tool in the following locations:

- `SOA_Oracle_Home\bpel\bin`

To use `schemac`, run a command similar to the following to generate the facades from WSDL or XSD files:

```
C:\BPEL_project_dir\> schemac *.wsdl /*.xsd
```

After you run `schemac`, it creates a `src` folder for a `HelperService.java` service and a `com` folder for the generated Java classes. Oracle provides a sample in the following directories that showcases the use of facade classes in Java bindings:

- `SOA_Oracle_Home\bpel\samples\tutorials\702.Bindings\JavaBinding`

When it generates the facade, `schemac` uses the following files:

- Using `build.xml`, `schemac` generates the source of the facade classes.
- The `schemac` tool creates a Java binding provider class `HelperService.java`, which in the `702.Bindings` example is located under `702.Bindings\JavaBinding\src\com\otn\services`. It has one method, which uses the facade classes `CommentsType` and `CommentType`:

```
public CommentsType addComment(CommentsType payload, CommentType comment)
```

- To map the XML types to the corresponding facade classes, a Java binding service is defined in the `HelperService.wsdl` file. See the `format:typeMapping` section of Java binding in [Example 18–3](#):

Example 18–3 Definition of Java Binding Service

```

<format:typeMapping encoding="Java" style="Java">
  <format:typeMap typeName="tns:commentType"
    formatType="com.otn.services.CommentType" />

```

```

    <format:typeMap typeName="tns:commentsType"
      formatType="com.otn.services.CommentsType" />
  </format:typeMapping>

```

18.1.2.3 bpelx:exec Built-in Methods

Table 18–1 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 18–1 Built in Methods for `bpelx:exec`

Method Name	Description
<code>Object lookup(String name)</code>	JNDI access
<code>Locator getLocator()</code>	BPEL Process Manager Locator
<code>long getInstanceId()</code>	Unique ID associated with each instance
<code>String setTitle(String title) / String getTitle()</code>	Title of this instance
<code>String setStatus(String status) / String getStatus()</code>	Status of this instance
<code>void setIndex(int i, String value) / String getIndex(int i)</code>	Six indexes can be used for search
<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>	
<code>Object getVariableData(String name, String part, String query)</code>	
<code>void setVariableData(String name, Object value)</code>	
<code>void setVariableData(String name, String part, Object value)</code>	
<code>void setVariableData(String name, String part, String query, Object value)</code>	

18.1.3 Using 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 application that invokes a service interface through a SOAP reference binding component. For this example, the service interface used is an Application Development Framework (ADF) business component (BC).

The high-level instructions for this scenario are as follows:

1. Create an ADF BC 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 ADF BC service WSDL into the SOA application.
4. Create a Web service binding to the ADF BC service interface.
5. Design a BPEL process in which you perform the following tasks:
 - a. Create a partner link for the ADF BC 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 ADF BC 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 ADF BC service to be remotely deployed.
7. Deploy the SOA application.
8. Invoke the SOA application from the Web service test page in Oracle Enterprise Manager 11g Application Server Control Console.

For more information, see the following:

- *Oracle Fusion Applications Developer Standards and Guidelines* for detailed instructions on creating this scenario
- *Application Development Framework Developer's Guide* for instructions on creating ADF BCs

18.2 Using Java Embedding in a BPEL Process

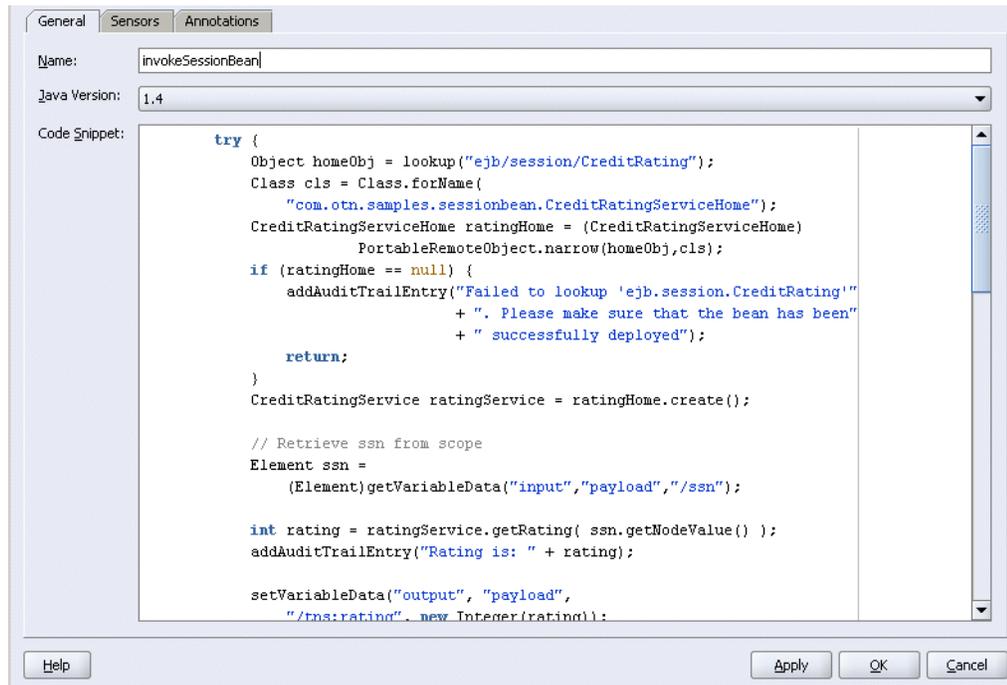
In Oracle JDeveloper, you can add the `bpelx:exec` activity, and copy the code snippet into a dialog box, as follows:

18.2.1 How To Use Java Embedding in a BPEL Process

1. Drag and drop the **Java Embedding** activity (with the coffee cup icon) from the **Component Palette**.
2. Double-click the **Java Embedding** activity to display the Java Embedding window.
3. Name the **Java Embedding** activity.
4. In the **Code Snippet** field, enter (or cut and paste) the Java code.

For example, the `bpel:exec` code example described under [Section 18.1.2.1, "Using the `bpel:exec` Tag to Embed Java Code Snippets into a BPEL Process"](#) on page 18-2 on page 18-2 appears as shown in [Figure 18-1](#):

Figure 18-1 *bpel:exec Code Example*



18.3 Using PL/SQL Procedures with SOA Applications

This section provides a high-level overview of how to use PL/SQL procedures with SOA applications.

- [Section 18.3.1, "How to Initiate a SOA Application with PL/SQL Procedures"](#)
- [Section 18.3.2, "How to Access a Service Implemented as a PL/SQL Procedure"](#)

18.3.1 How to Initiate a SOA Application with PL/SQL Procedures

PL/SQL stored procedures can initiate a BPEL process in a SOA application by raising an event through the event delivery network-database (EDN-DB). In the SOA application, a mediator component can subscribe to the raised event and route it as required.

The high-level instructions for this scenario are as follows:

1. Create a SOA application with a mediator component that subscribes to an event. Note the following:
 - The business event filter can be set by business event name or by specifying an XPath expression on the event payload.
 - The event definition language (EDL) for the business event can either be supplied in the `composite.xml` file of the SOA application or deployed separately in a MAR.
2. Create a BPEL component.
3. Connect (wire) the mediator component reference (right side) to the BPEL component service (left side).
4. Complete and deploy the SOA application.
5. Call the EDN-DB API method `publish_event` from a PL/SQL stored procedure with the event namespace and the event payload as an `xmltype`.

Note: Business event publishing is an asynchronous action only.

For more information, see the following:

- [Chapter 8, "Business Events and the Event Delivery Network"](#)
- *Oracle Fusion Applications Developer Standards and Guidelines* for detailed instructions on creating this scenario

18.3.2 How to Access a Service Implemented as a PL/SQL Procedure

You can create SOA applications that access logic implemented as PL/SQL in a database. For instance, a BPEL process may need to orchestrate a variety of services, including PL/SQL stored procedures. A recommended method for performing this is to place an ADF BC in front of the PL/SQL code that is then accessed over SOAP by the SOA application.

The high-level instructions for this scenario are as follows:

1. Create an ADF BC service.
2. Write a method in your application module that accesses the PL/SQL stored procedure by following the instructions in *Oracle Application Developer Framework Developer's Guide for Forms/4GL Developers*.
3. Generate the service interface for the ADF BC. This action generates the WSDL and XSD files for the service.
4. Invoke the service method through a SOAP binding as described in [Section 18.1.3, "Using Java Code Wrapped in a Service Interface"](#) on page 18-5.

For more information about creating this scenario, see *Oracle Fusion Applications Developer Standards and Guidelines*.

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 19.1, "Introduction to Event and Timeout Concepts"](#)
- [Section 19.2, "Adding a Pick Activity to Select Between Continuing a Process or Waiting"](#)
- [Section 19.3, "Adding a Wait Activity to Set an Expiration Time"](#)
- [Section 19.4, "Setting Timeouts for Synchronous Processes"](#)
- [Section 19.5, "Use Case for Events and Timeouts"](#)

19.1 Introduction to Event and Timeout Concepts

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.

19.2 Adding 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 19–1](#) provides an overview. The following activities take place:

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 the same as the code for receiving a response from the Star Loan Web service before a timeout was added.

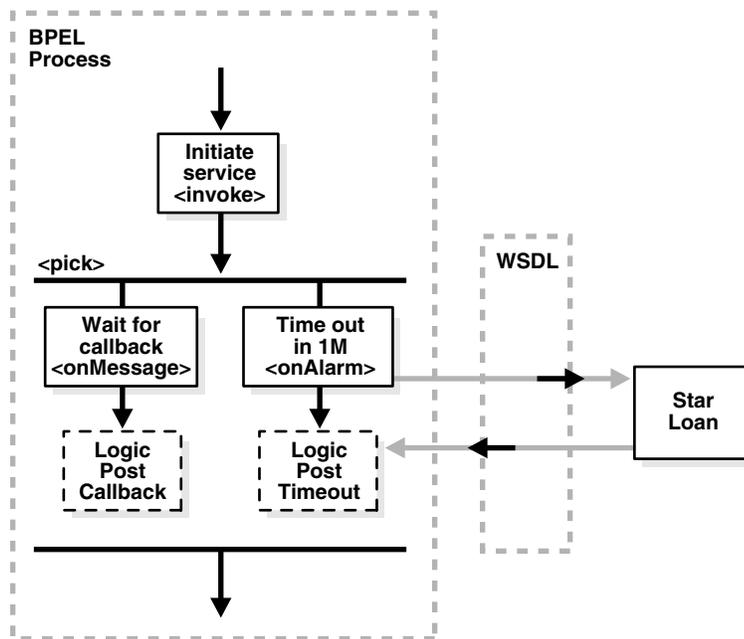
- onAlarm:** This condition has code for a timeout of one minute. This time is defined as PT1M, which means to wait one minute before timing out. In this timeout setting, S stands for seconds, M for one minute, H for hour, D for day, and Y for year. In the unlikely event that you want a time limit of 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.

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>

- The pick activity condition that completes first is the one that the BPEL process service component executes. The other branch then is not executed.

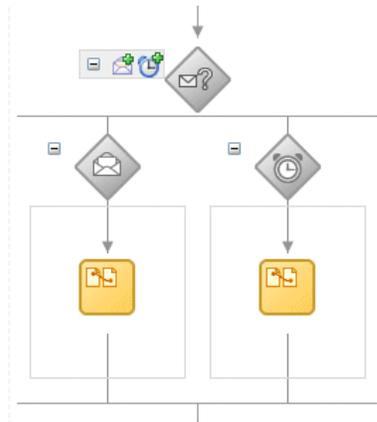
Figure 19–1 Overview of the Pick Activity



19.2.1 How To Add a Pick Activity

- In the SOA Composite Editor, double-click the BPEL process service component.
- From the Component Palette, drag a **Pick** activity into the designer.
- Expand the **Pick** activity.

The **Pick** activity includes the **onMessage** (envelope icon) and **onAlarm** (alarm clock icon) branches. [Figure 19–2](#) provides an example.

Figure 19–2 Pick Activity

4. Double-click the **OnAlarm** branch of the **pick** activity shown and set its time limit to 1 minute instead of 1 hour. [Figure 19–3](#) provides an example.

Figure 19–3 OnAlarm Branch

5. Click **OK**.
6. Double-click the **onMessage** branch, and edit its attributes to receive the response from the loan service. [Figure 19–4](#) provides an example.

Figure 19–4 *onMessage Branch*

19.2.2 What Happens When You Add a Pick Activity

The code segment in [Example 19–1](#) defines the `pick` activity for this operation:

Example 19–1 *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 timesout -->
  <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>

```

For more information, see the *SOA_Oracle_Home\bpel\samples\references\Pick* sample.

19.3 Adding 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.

For more information, see *SOA_Oracle_Home\bpel\samples\references\Wait* for examples of defining a wait activity.

19.3.1 How To Add a Wait Activity

1. In the SOA Composite Editor, double-click the BPEL process service component.
2. From the Component Palette, drag a **Wait** activity into the designer.
- 3.

19.3.2 What Happens When You Add a Wait Activity

Exactly one of the expiration criteria must be specified, as shown in [Example 19-2](#).

Example 19-2 Wait Activity

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
  standard-elements
</wait>
```

19.4 Setting Timeouts for Synchronous Processes

For synchronous processes that connect to a remote database, you must increase the `syncMaxWaitTime` timeout property in the

19.4.1 How To Set Timeouts for Synchronous Processes

1. Open the *ORACLE_HOME\j2ee\home\applications\soa-infra\configuration\bpel-config.xml* file.
2. Edit the value for the `syncMaxWaitTime` property. [Example 19-3](#) provides an example.

Example 19-3 syncMaxWaitTime timeout property

```
<property id="syncMaxWaitTime">
  <name>Delivery result receiver maximum wait time</name>
  <value>45</value>
  <comment>
    <![CDATA[The maximum time the process result receiver will wait for a
    result before returning. Results from asynchronous BPEL processes are
    retrieved synchronously via a receiver that will wait for a result from the
    container.
    <p/>
    The default value is 45 seconds.]]>
  </comment>
</property>
```

19.5 Use Case for Events and Timeouts

In this use case, you program a BPEL process service component to wait one minute for a response from the Star Loan Web service. If Star Loan does not respond in one minute, then the BPEL process service component automatically selects the United Loan offer. 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.

Note: The sample demonstrated in `108.Timeouts` is not included in the `soa-samples.zip` file for beta 3.

Invoking a BPEL Process Service Component

This chapter shows how to invoke a BPEL process service component to perform functions or use services.

This chapter contains the following topics:

- [Section 20.1, "Use Case for Invoking a BPEL Process Service Component"](#)
- [Section 20.2, "Sending Messages to a BPEL Process Service Component"](#)

20.1 Use Case for Invoking a BPEL Process Service Component

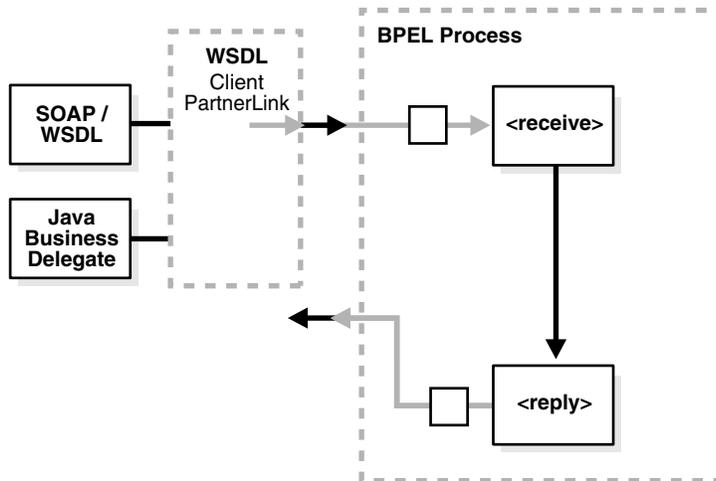
In this use case, you learn how to invoke synchronous and asynchronous BPEL processes through the simple object access protocol (SOAP). The BPEL process service component accepts a social security number and sends a credit rating in return.

Note: The sample demonstrated in `102.InvokingProcesses` is not included in the `soa-samples.zip` file for beta 3.

20.2 Sending Messages to a BPEL Process Service Component

You can invoke a BPEL process service component as a Web service through a WSDL or SOAP interface. The application puts the request in the form of a payload that then goes to the BPEL process service component. The BPEL process service component receives the payload and responds with a payload containing the information that the application requested.

[Figure 20-1](#) shows how an application interacts with a BPEL process service component through a client partner link, using one of a number of possible protocols.

Figure 20–1 Application Interaction with a BPEL Process Service Component

20.2.1 Invoking a BPEL Process Service Component with the Web Service/SOAP Interface

After you deploy a BPEL process service component to Oracle BPEL Server, it is automatically published as a Web service. This means that the process can be accessed through its XML/SOAP/WSDL interface without any additional developer effort. Supporting a standard Web services interface means that BPEL processes can be invoked from any client technology that supports Web services. This includes Microsoft .NET, Sun's JAX-RPC implementation, Apache Axis, Oracle JDeveloper, and many other Web services tool kits. In addition, it means that BPEL and Oracle BPEL Process Manager can publish Web services. Those services, both synchronous and asynchronous, can be invoked from applications and services implemented with nearly any technology and language.

You access a BPEL process service component through its Web service interface in the standard way you access any Web service: by writing a client that uses the BPEL process service component WSDL interface definition and SOAP as a protocol.

Coordinating Master and Detail Processes

This chapter describes master and detail process coordinations. 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 21.1, "Introduction to Master and Detail Process Coordinations"](#)
- [Section 21.2, "Defining Master and Detail Process Coordination in Oracle JDeveloper"](#)

For more information about master and detail coordinations, see the sample located in the `bpel\tutorials\T116_MasterDetailApp` directory of the `soa-samples.zip` file

21.1 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.

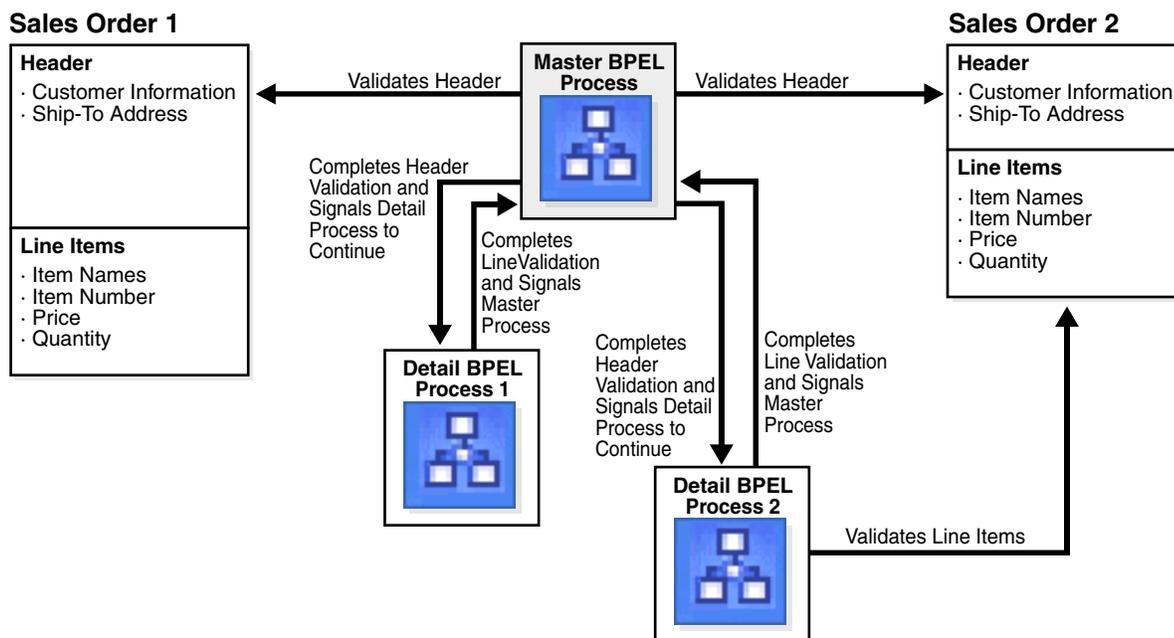
Potential coordination points are as follows:

- 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 21–1 provides an overview of the header and line item validation coordination points between one master process and two detail processes.

Figure 21–1 Master and Detail Coordination Overview (One BPEL Process to Two Detail Processes)

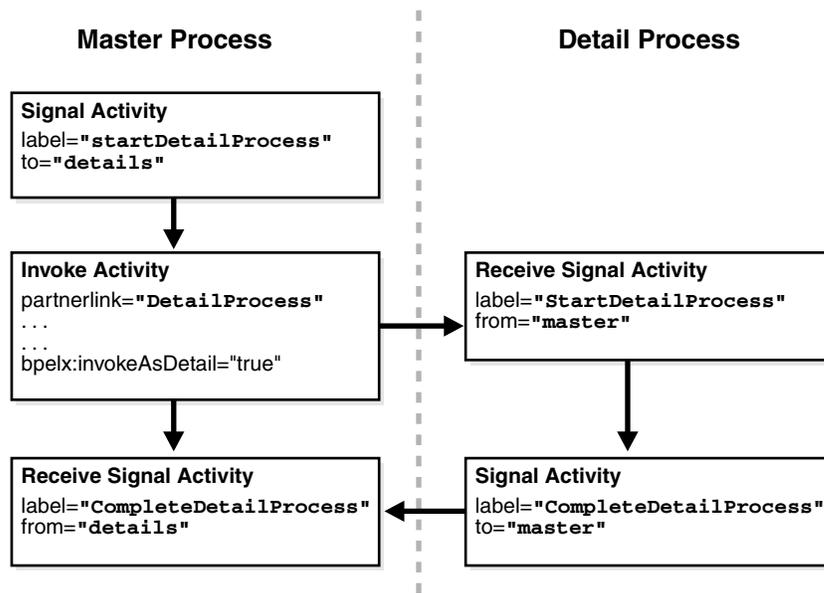


The following BPEL process activities coordinate actions between the master and detail processes:

- **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 21–2 provides an overview of the BPEL process flow coordination.

Figure 21–2 Master and Detail Syntax Overview (One BPEL Process to One Detail Process)

As shown in [Figure 21–2](#), each master and detail process includes a signal and receive signal activity. [Table 21–1](#) describes activity responsibilities based on the type of process in which they are defined.

Table 21–1 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.

21.1.1 BPEL File Definition for the Master Process

The BPEL process 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 four detail processes. [Example 21–1](#) provides an example.

Example 21–1 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 four 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 four 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 21–2](#) provides an example.

Example 21–2 `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 21–3](#) provides an example.

Example 21–3 Receive Signal Activity

```
<bpelx:receiveSignal name="waitForNotifyFromDetailProcess"
  label="detailProcessComplete"
  from="details"/>
```

21.1.1.1 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 21–4](#) provides an example.

Example 21–4 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 21–5](#) provides an example.

Example 21–5 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 21–6 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 21–7](#) waits for a return signal from detail process `DetailProcess0`.

Example 21–7 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 21–8](#) also waits for a return signal from `DetailProcess1` and `DetailProcess2`.

Example 21–8 Second Receive Signal Activity

```
<!-- This is a receiveSignal waiting for 2 child (detail) process 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.

21.1.2 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 21–9](#) 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 21–9 startDetailProcess Label Value

```
<bpelx:receiveSignal name="waitForNotifyFromMasterProcess"
  label="startDetailProcess" from="master"/>
```

A signal activity indicates that the detail process shown in [Example 21–10](#) 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 21–10 Signal Activity

```
<bpelx:signal name="notifyMasterProcess" label="detailProcessComplete"
  to="master"/>
```

21.2 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 JDeveloper. In this example, one master process and one detail process are defined.

- [Section 21.2.1, "How to Create a Master Process"](#)
- [Section 21.2.2, "How to Create a Detail Process"](#)
- [Section 21.2.3, "How to Create an Invoke Activity"](#)

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.

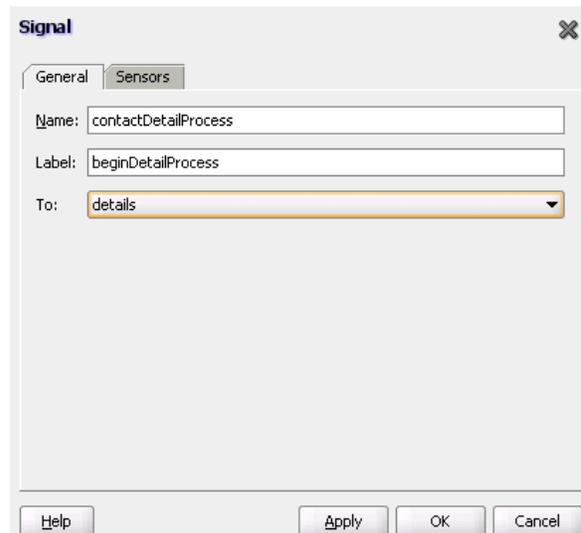
21.2.1 How to Create a Master Process

1. Create a BPEL process service component in the SOA Composite Editor. For this example, the process is named **MasterProcess**.
2. Double-click the **MasterProcess** BPEL process.
3. Expand **BPEL Activities** in the **Component Palette**.
4. Drag and drop a **Signal** activity into your BPEL process service component.
5. Double-click the **Signal** activity.
This activity signals the detail process to perform processing at runtime.
6. Enter the following details:

Field	Value
Name	Enter a name (for this example, contactDetailProcess).
Label	Enter a label name (for this example, beginDetailProcess). This label must match the receive signal activity label you set in the detail process in Step 5 on page 21-8.
To	Select details as the type of process to receive this signal.

Figure 21–3 shows the Signal dialog.

Figure 21–3 Signal Dialog



7. Click **OK**.
8. Drag and drop a **Receive Signal** activity into your BPEL process service component.
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 following details:

Field	Value
Name	Enter a name (for this example, waitForDetailProcess).
Label	Enter a label name (for this example, completeDetailProcess). This label must match the signal activity label you set in the detail process in Step 9 on page 21-9.
To	Select details as the type of process from which to receive the signal.

Figure 21-4 shows the Receive Signal dialog.

Figure 21-4 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.

21.2.2 How to Create a Detail Process

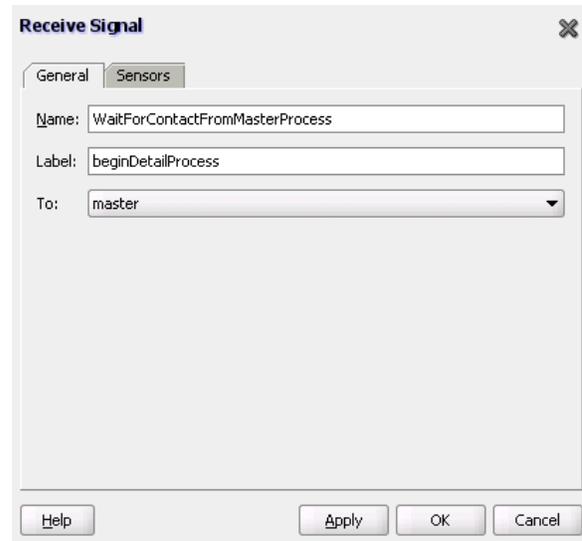
1. Create a second BPEL process service component in the SOA Composite Editor. For this example, the process is named **DetailProcess**.
2. Double-click the **DetailProcess** BPEL process.
3. Drag and drop a **Receive Signal** activity into your BPEL process service component.
4. Double-click the **Receive Signal** activity.
This activity enables the detail process to wait until it receives the signal executed by its master process.
5. Enter the following details:

Field	Value
Name	Enter a name (for this example, WaitForContactFromMasterProcess).

Field	Value
Label	Enter a label name (for this example, beginDetailProcess). This label must match the signal activity label you set in the master process in Step 6 on page 21-7.
To	Select master as the type of process from which to receive the signal.

Figure 21-5 shows the Receive Signal dialog.

Figure 21-5 Receive Signal Dialog



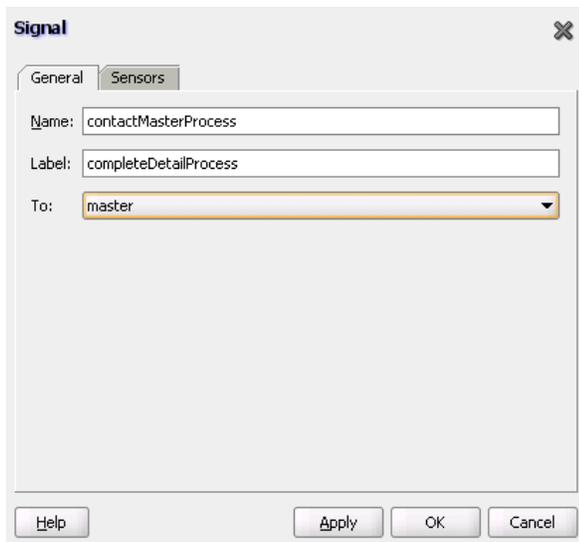
6. Click **OK**.
7. Drag and drop a **Signal** activity into your BPEL process service component.
8. Double-click the **Signal** activity.

This activity enables the detail process to signal its associated master process at runtime that processing is complete.

9. Enter the following details:

Field	Value
Name	Enter a name (for this example, contactMasterProcess).
Label	Enter a label name (for this example, completeDetailProcess). This label must match the receive signal activity label you set in the master process in Step 10 on page 21-7.
To	Select master as the destination.

Figure 21-6 shows the Signal dialog.

Figure 21–6 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.

21.2.3 How to Create an Invoke Activity

1. Return to the MasterProcess master process.
2. Drag and drop 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 21-8 as the partner link.
5. Complete all remaining fields in the Invoke window, and click **OK**.
6. Click **Source** in the BPEL process diagram window.
7. Add `bpelx:invokeasdetail` to the invoke activity and set it to `true`, as shown in [Example 21–11](#).

Example 21–11 `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 21–12](#).

Example 21–12 *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 21–13](#).

Example 21–13 *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. Select **Save All** from the **File** main menu.

Master and detail coordination design is now complete.

Interaction Patterns in BPEL Processes

This chapter identifies common interaction patterns between a BPEL process service component and another application, and shows the best use practices for each.

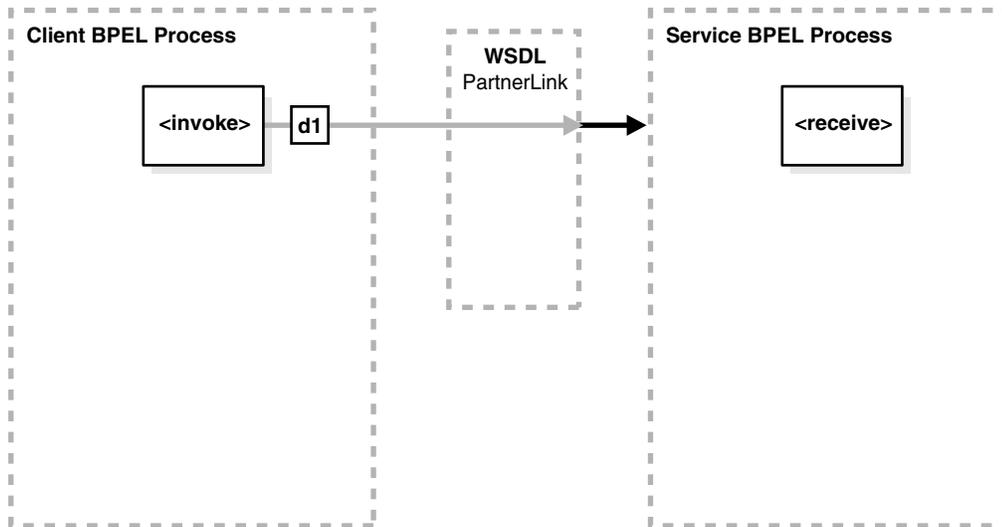
This chapter includes the following sections:

- [Section 22.1, "One-Way Message"](#)
- [Section 22.2, "Synchronous Interaction"](#)
- [Section 22.3, "Asynchronous Interaction"](#)
- [Section 22.4, "Asynchronous Interaction with Timeout"](#)
- [Section 22.5, "Asynchronous Interaction with a Notification Timer"](#)
- [Section 22.6, "One Request, Multiple Responses"](#)
- [Section 22.7, "One Request, One of Two Possible Responses"](#)
- [Section 22.8, "One Request, a Mandatory Response, and an Optional Response"](#)
- [Section 22.9, "Partial Processing"](#)
- [Section 22.10, "Multiple Application Interactions"](#)
- [Section 22.11, "Summary"](#)

22.1 One-Way Message

In a one-way message, or fire and forget, the client sends a message to the service, and the service does not need to reply. [Figure 22-1](#) provides an overview.

Figure 22–1 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 WSDL file defines the interaction.

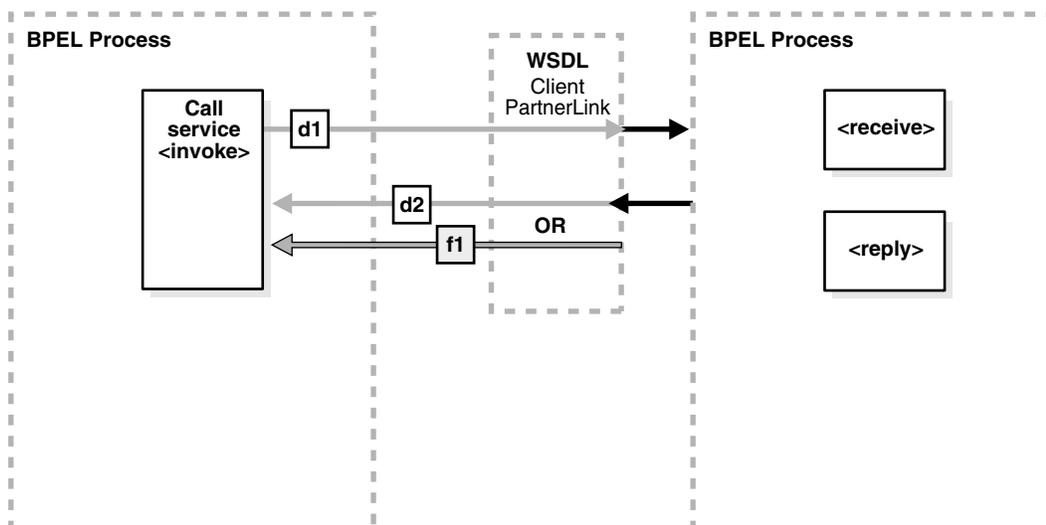
BPEL Process Service Component as the Service

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

22.2 Synchronous Interaction

In a synchronous interaction, a client sends a request to a service, and receives an immediate reply. The 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. [Figure 22–2](#) provides an overview.

Figure 22–2 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

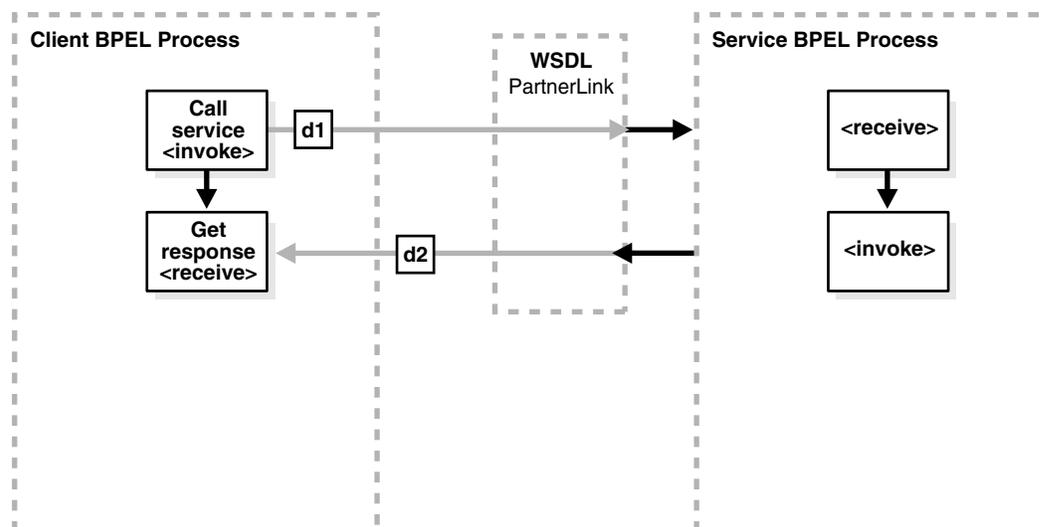
When the BPEL process service component is on the service side of a synchronous transaction, it needs a receive activity to accept the incoming request, and a reply activity to return either the requested information or an error message (a fault).

See Also: [Chapter 13, "Invoking a Synchronous Web Service from a BPEL Process"](#)

22.3 Asynchronous Interaction

In an asynchronous interaction, a client sends a request to a service and waits until the service replies. [Figure 22-3](#) provides an overview.

Figure 22-3 *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

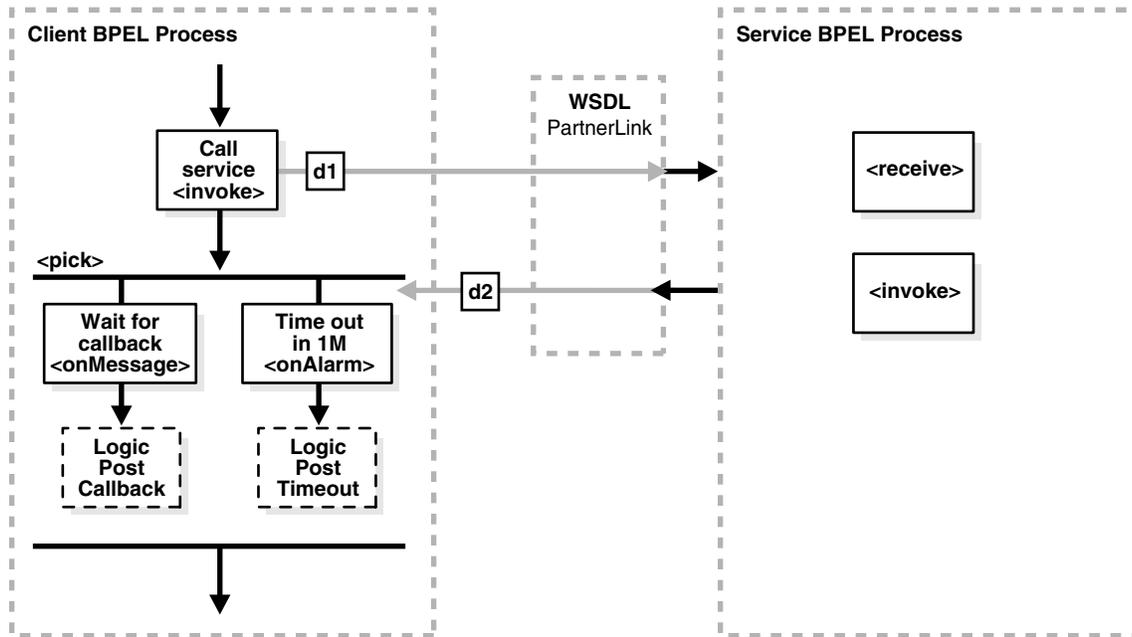
As with a synchronous transaction, when the BPEL process service component is on the service side of an asynchronous transaction, it needs a receive activity to accept the incoming request and an invoke activity to return either the requested information or a fault.

See Also: [Chapter 14, "Invoking an Asynchronous Web Service from a BPEL Process"](#)

22.4 Asynchronous Interaction with Timeout

In an asynchronous interaction with a timeout, 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. [Figure 22-4](#) provides an overview.

Figure 22-4 *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.

See Also: [Section 19.2, "Adding a Pick Activity to Select Between Continuing a Process or Waiting"](#) on page 19-1

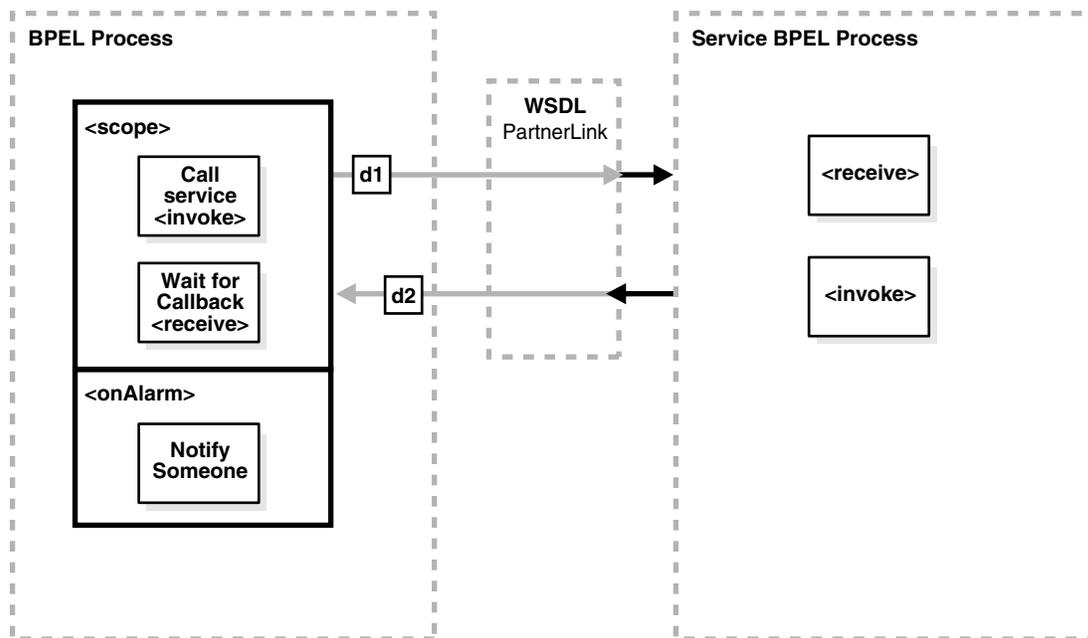
BPEL Process Service Component as the Service

The behavior of the service BPEL process service component is the same as with the asynchronous interaction with the BPEL process service component as the service, as described in ["BPEL Process Service Component as the Service"](#) on page 22-3.

22.5 Asynchronous Interaction 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 22-5](#) provides an overview.

Figure 22–5 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.

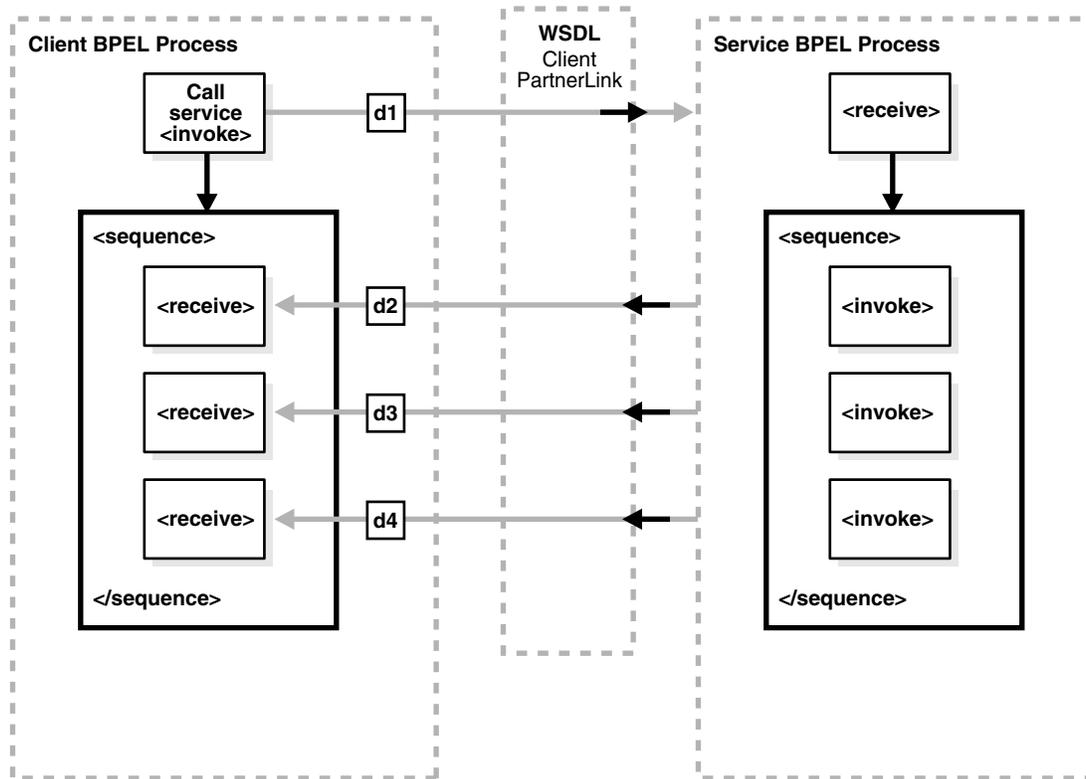
BPEL Process Service Component as the Service

The behavior for the service BPEL process service component is the same as with the asynchronous interaction with the BPEL process service component as the service, as described in "BPEL Process Service Component as the Service" on page 22-3.

22.6 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 22–6 provides an overview.

Figure 22–6 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.

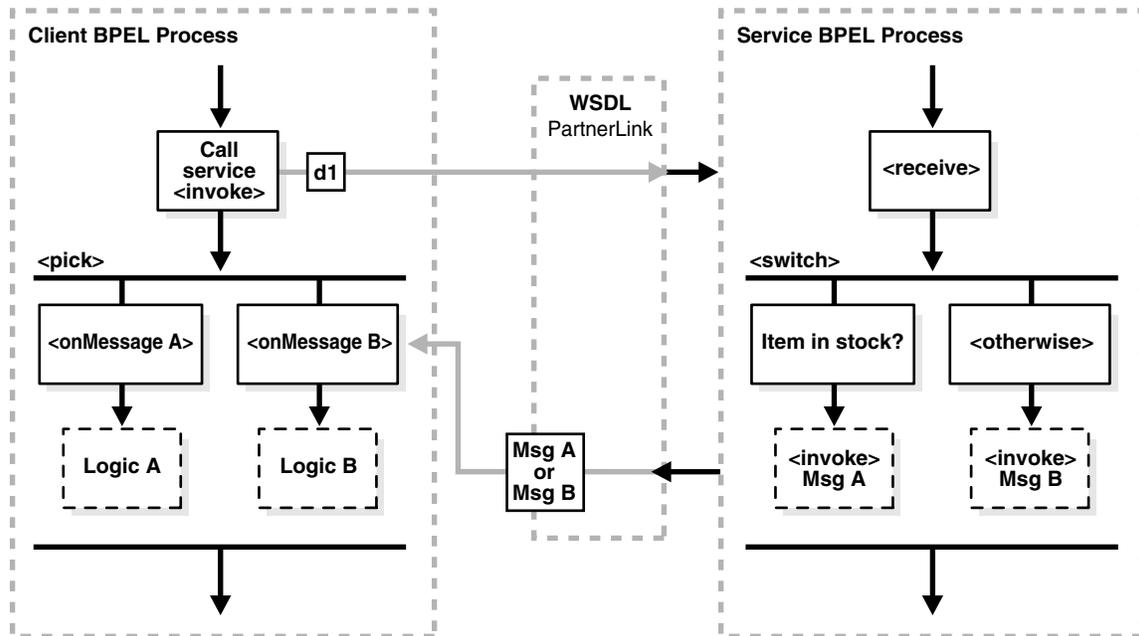
BPEL Process Service Component as the Service

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.

22.7 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 22–7](#) provides an overview.

Figure 22–7 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.

See Also: [Section 19.2, "Adding a Pick Activity to Select Between Continuing a Process or Waiting"](#) on page 19-1

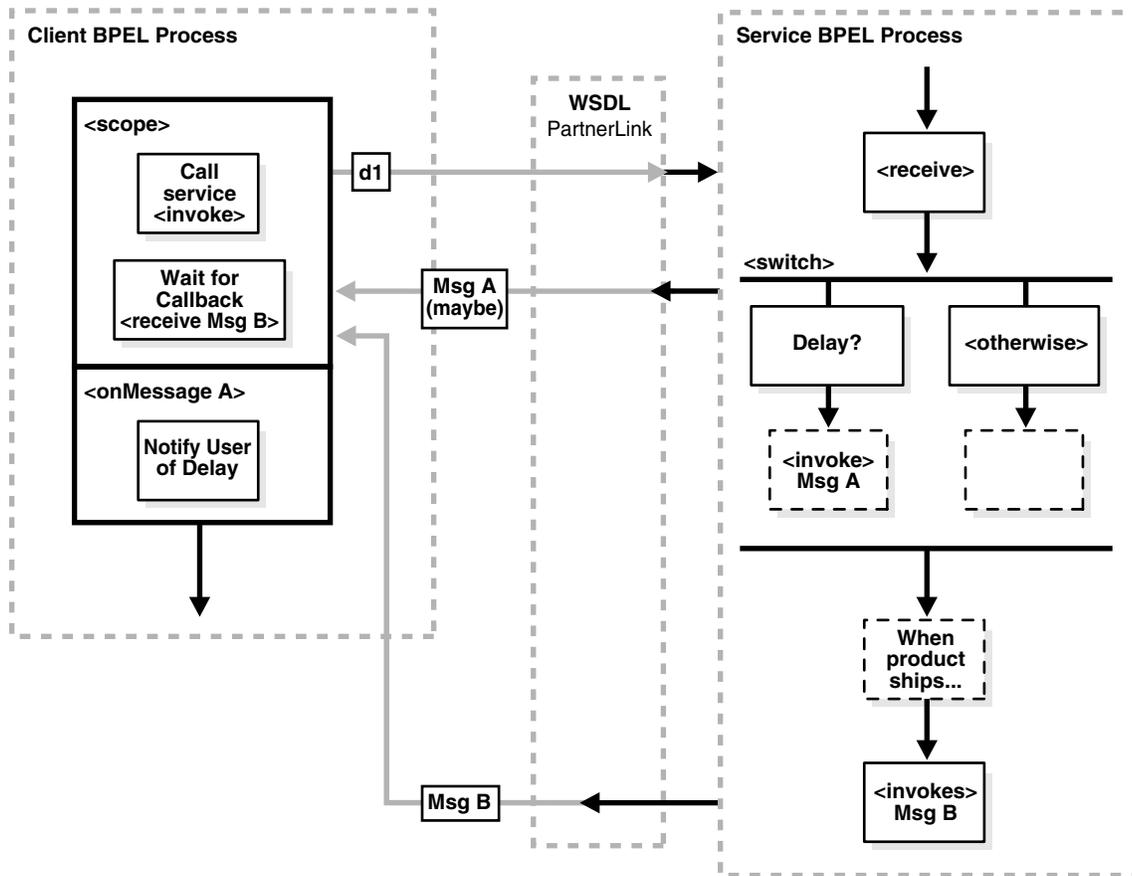
BPEL Process Service Component as the Service

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.

22.8 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 22–8](#) provides an overview.

Figure 22–8 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

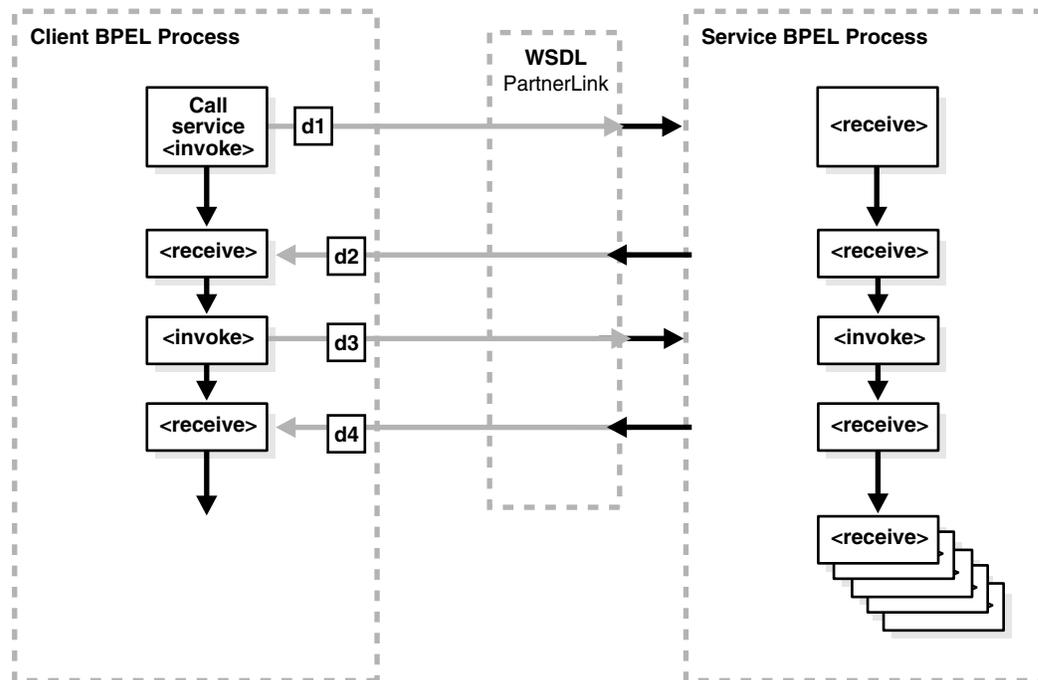
The BPEL service needs a scope activity containing the receive activity and an invoke activity to send the mandatory shipping message, and the scope’s onAlarm handler to send the optional delayed message if a timer expires (for example, send the delayed message if the item is not shipped in 24 hours).

22.9 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 22–9](#) provides an overview.

Figure 22–9 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

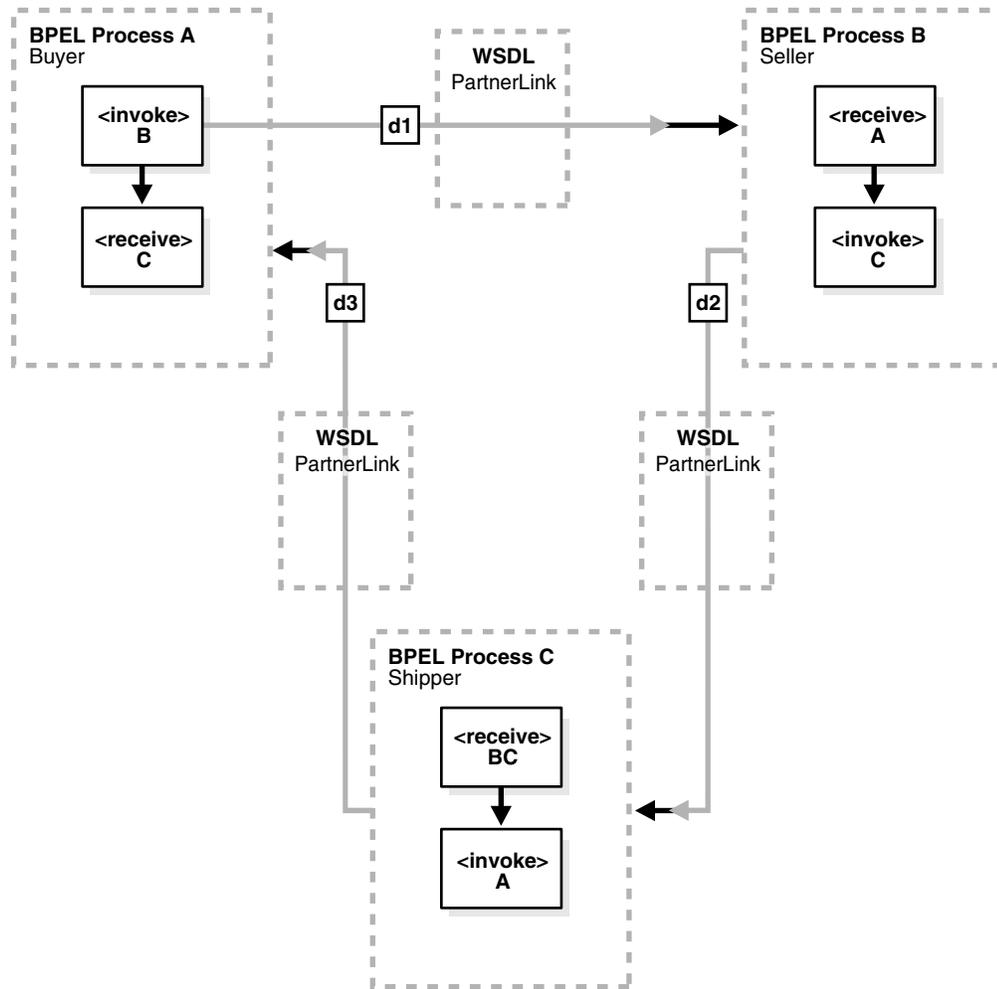
The BPEL service needs a receive activity for each request from the client, and a reply activity for each response. Once the responses are finished, the service BPEL process service component can continue with its processing, using the information gathered in the interaction to perform the necessary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

22.10 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 once. Therefore, a mechanism is required for keeping track of which message goes where. [Figure 22–10](#) provides an overview.

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

Figure 22-10 Multiple Party Interactions



See Also: [Chapter 14, "Invoking an Asynchronous Web Service from a BPEL Process"](#) for more information about WS-addressing and correlation sets

22.11 Summary

BPEL process service components can serve as both clients or services, and this chapter lists several common interaction patterns and describes best practices for implementing these interactions.

Notifications and the Oracle User Messaging Service

The Oracle User Messaging Service in Oracle SOA Suite enables you to send notifications from a BPEL process using a variety of channels. These notifications are delivered by e-mail, voice message, instant messaging (IM), or short message service (SMS).

This chapter contains the following topics:

- [Section 23.1, "Use Cases for Notifications"](#)
- [Section 23.2, "Introduction to Oracle User Messaging Service and Notification Concepts"](#)
- [Section 23.3, "Configuring the Service in Oracle JDeveloper"](#)
- [Section 23.4, "Summary"](#)

23.1 Use Cases for Notifications

Various scenarios may require sending e-mail 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 or human task can use the Oracle User Messaging Service to alert users by voice, IM, SMS, or e-mail. In an approval workflow (for example, an expense report approval), you can send notifications to the task assignee when a specific task requires action, or you can notify the task creator by e-mail when the approval is complete. In some cases, contact information (e-mail address or telephone number) is obtained dynamically as part of the process and in other cases the details are looked up from a user directory.

The tutorial `130.SendEmailWithAttachments` demonstrates how to model a notification in Oracle JDeveloper and send an e-mail with an attachment.

Note: The sample demonstrated in `130.SendEmailWithAttachments` is not included in the `soa-samples.zip` file for beta 3.

23.2 Introduction to Oracle User Messaging Service and Notification Concepts

Terms used include:

- Notification—an asynchronous message sent to a user by a specific channel. The message can be sent as an e-mail message, a voice message, IM message, or an SMS message.
- Actionable notification—a notification to which the user can respond. For example, workflow sends an e-mail or IM message to a manager to approve or reject a purchase order. The manager approves or rejects the request by replying to the e-mail or IM with appropriate content.
- Human task e-mail notification layer—sends e-mail 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 e-mail 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 e-mail 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).

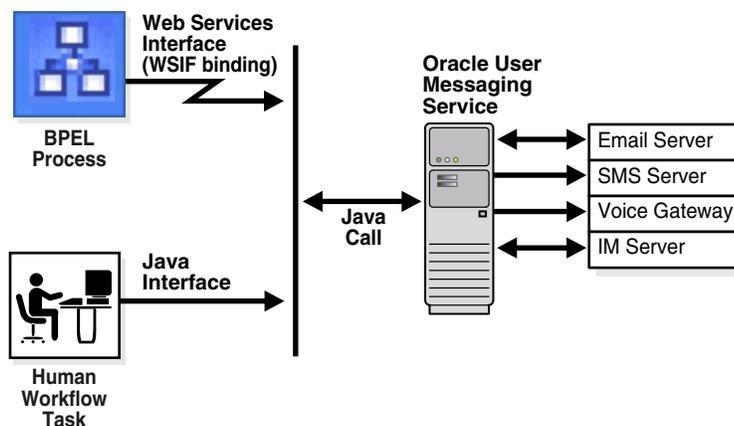
You can send e-mail through the SMTP protocol and receive e-mail from IMAP- and POP-based e-mail accounts.

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 —works with the human task e-mail notification layer to improve message reliability and handles the sending and receiving of messages to and from devices, such as IM, SMS, or voice. The Oracle User Messaging Service is a new feature for release 11g. Oracle BPEL Process Manager is preconfigured to send notifications using the Oracle User Messaging Service.

Figure 23–1 shows the Oracle User Messaging Service interfaces and supported service types.

Figure 23–1 Service Interfaces and Supported Service Types



See Also:

- [Section 28.2, "Notifications from Human Workflow"](#) on page 28-20
- [Section 26.6.7, "Specifying Participant Notification Preferences"](#) on page 26-53 for instructions on specifying e-mail notifications in the Human Task editor

23.2.1 Reliable Notifications

Oracle BPEL Process Manager provides support for the reliable notifications. The outbound notification creates a notification message with a unique notification ID and stores the message and unique ID in the dehydration store. It then enqueues this unique ID in the JMS queue and commits the transaction. A message driven-bean (MDB) listening on this queue dequeues the message and sends a notification to the user. If there is any notification failure, the notification retries three times. If the retries all fail, it marks this notification as errored.

To send an error notification after resolving the problem, you must write a script to update the BPELNotification table status to SEND. For example:

```
UPDATE BPELNotification
   SET status = 'RETRY',
       ATTEMPTEDNUMBER = 0
  WHERE ID = <notification id>
```

By default, the notification retries three time. If you want to add more retries (for example, 5), add the following property in `SOA_ORACLE_HOME\j2ee\home\applications\soa-infra\configuration\workflow-config.xml` and restart Oracle BPEL Server:

```
<property name="oracle.bpel.services.notification.maxattempt" value="5" />
```

The notification thread that is running tries to send the notification every 15 minutes. You can change this interval by adding the following property in `workflow-config.xml`. For example, to retry every 10 minutes:

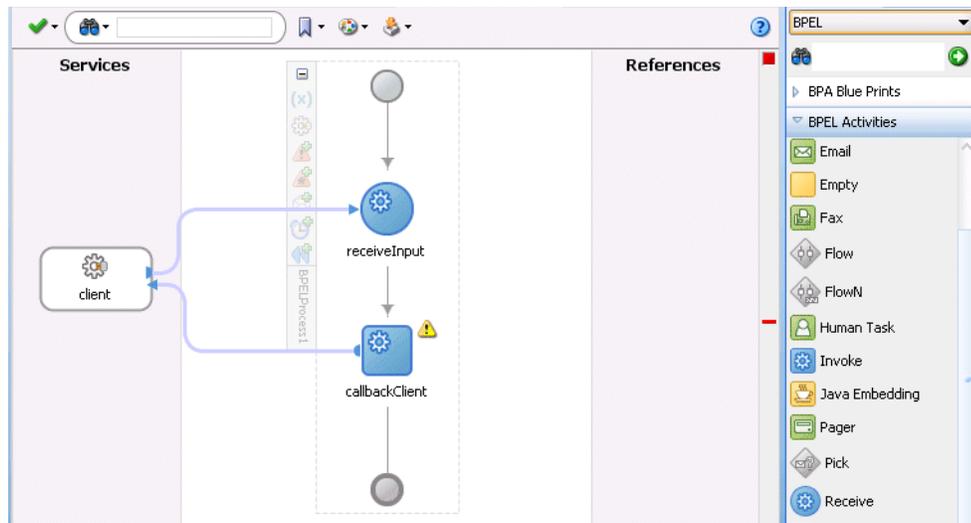
```
<property name="oracle.bpel.services.notification.publisher_interval" value="10" />
```

See Also: [Section 28.2.8, "Reliability Support"](#) on page 28-27

23.3 Configuring the Service in Oracle JDeveloper

The diagram window in Oracle JDeveloper includes the notification channels in the **Component Palette**, as shown in [Figure 23-2](#).

Figure 23–2 Diagram Window in Oracle JDeveloper—Notification Channel



To use a notification channel, do the following:

1. Select **BPEL** from the **Component Palette** list.
2. Expand **BPEL Activities**.
3. Drag and drop a notification channel from the **Component Palette** list:
 - **Email**
 - **SMS**
 - **Voice**
4. See the following section based on the notification channel you selected.

If You Selected...	See...
Email	Section 23.3.1, "The E-mail Notification Channel" on page 23-5 to configure e-mail notification
SMS	Section 23.3.2, "The SMS Notification Channel" on page 23-9 to configure SMS notification
Voice	Section 23.3.3, "The Voice Notification Channel" on page 23-10 to configure voice message notification

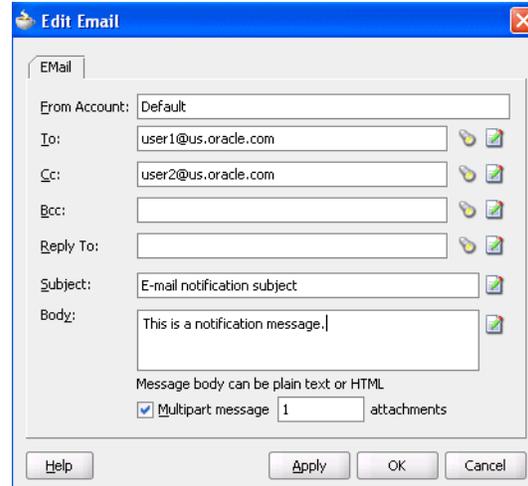
Notes:

- After configuring Oracle User Messaging Service for e-mail and other channels in Oracle JDeveloper, set the `NotificationMode` parameter to either `ALL` or `EMAIL` in the `workflow-notification-config.xml` file, as described in [Section 28.2.1, "Configuring the Notification Channel"](#) on page 28-21.
- There is currently no support for the IM notification channel in Oracle JDeveloper. IM can be used for sending actionable messages. See [Section 28.2.4.2, "Sending Actionable Instant Messages"](#) on page 28-26 for configuration instructions.

23.3.1 The E-mail Notification Channel

When you select **Email** from the **Component Palette**, the Edit Email window appears. [Figure 23–3](#) shows the required e-mail notification parameters.

Figure 23–3 Edit Email Window



1. Enter information for each field as described in [Table 23–1](#).

Table 23–1 E-mail Notification Parameters

Name	Description
From Account	The name of the account used to send this message. The configuration details for this e-mail account name must exist on Oracle BPEL Server.
To	The e-mail address to which the message is to be delivered. This can be <i>a</i>) a static e-mail address entered at the time the message is created, or <i>b</i>) an e-mail address looked up using the identity service, or <i>c</i>) a dynamic address from the payload. The XPath Expression Builder can be used to get the dynamic e-mail address from the input. See Section 23.3.4, "Setting E-mail Addresses and Telephone Numbers Dynamically" on page 23-11.
CC and Bcc	The e-mail addresses to which the message is copied and blind copied. This can be a static or dynamic address as described for the To address.
Reply To	The e-mail address to use for replies. This can be a static or dynamic address as described for the To address.
Subject	Subject of the e-mail message. This can be free 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	Message body of the e-mail message. This can be plain text, XML, free text, or dynamic text, as described for the Subject parameter.
Multipart message with <i>n</i> attachments	Select to specify e-mail attachments. See Section 23.3.1.1, "Setting E-mail Attachments" on page 23-6. The number of attachments if Multipart message is selected. The number includes the body. For example, if you have a body and one attachment, specify 2 here.

2. Click **OK**.

The BPEL fragment that invokes the Oracle User Messaging Service to send the e-mail message is created.

See Also: *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details about e-mail configuration instructions to perform outside of Oracle JDeveloper

23.3.1.1 Setting E-mail Attachments

When you send e-mail attachments, you mark the e-mail as a multipart message and set the number of attachments to send. The number of attachments includes the body plus the attachments. (For example, to send an e-mail message with one file as an attachment, set the number to 2.) 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 one attachment to an e-mail message, do the following:

1. Select **Email** as the notification channel from the **Component Palette**.
2. Specify values for **To**, **Subject**, and **Body**.
3. Select **Multipart message** and enter 2 for the number of attachments. (Note that the number of attachments must include the body part.)

The `MultiPart` element with two body parts is generated. The first body part is for the message body and the other is used for the attachment. The BPEL fragment with an `assign` activity with multiple `copy` rules is generated. One of the `copy` rules copies the attachment, as follows:

```
<assign name="Assign">
  <copy>
    <from expression="string('Default')"/>
    <to variable="varNotificationReq" part="EmailPayload"
      query="/EmailPayload/ns1:FromAccountName"/>
    </copy>
  ...
  <!-- copy statements relate to body and attachment -->
  <copy>
    <from>
      <Content xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
        <MimeType
          xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">multipart/mixed
        </MimeType>
        <ContentBody
          xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
          <MultiPart
            xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
            <BodyPart
              xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
              <MimeType
                xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
              <ContentBody
                xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
              <BodyPartName
                xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
            </BodyPart>
          </BodyPart>
        </BodyPart>
      </Content>
    </from>
  </copy>
</assign>
```

```

        <MimeType
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
        <ContentBody
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
        <BodyPartName
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
        </BodyPart>
        </MultiPart>
        </ContentBody>
    </Content>
</from>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content"/>
</copy>
<copy>
    <from expression="string('text/html')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:
MultiPart/ns1:BodyPart[1]/
ns1:MimeType"/>
</copy>
<copy>
    <from expression="string('NotificationAttachment1.html')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:
MultiPart/ns1:BodyPart[1]/ns1:BodyPartName"/>
</copy>
<copy>
    <from expression="string('This is a test message from John Cooper')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:
MultiPart/ns1:BodyPart[1]/
ns1:ContentBody"/>
</copy>
<copy>
    <from expression="string('text/html')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:
MultiPart/ns1:BodyPart[2]/
ns1:MimeType"/>
</copy>
<copy>
    <from expression="string('NotificationAttachment2.html')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:
MultiPart/ns1:BodyPart[2]/
ns1:BodyPartName"/>
</copy>
<copy>
    <from expression="string('message2')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:
MultiPart/ns1:BodyPart[2]/
ns1:ContentBody"/>
</copy>
</assign>

```

4. Search for `BodyPart [2]` and set the required values. The steps to send the attachment `MyImage.gif`, for example, are as follows:

- a. Search for `BodyPart[2] MimeType` and change the from expression to copy 'image/gif' as the MimeType (instead of the autogenerated 'text/html').
- b. Search for `BodyPart[2] BodyPartName` and change the from expression to copy 'MyImage.gif' (instead of the autogenerated 'NotificationAttachment2.html').
- c. Search for `BodyPart[2] ContentBody` and change the from expression to copy the content of `MyImage.gif` (instead of the autogenerated expression `string('message2')`).

You can use the `readFile XPath` function to read the contents of the file:

```
ora:readFile('<name of the file in the project | HTTP URL | File URL>')
```

Examples:

```
ora:readFile('MyImage.gif') will read the file from the bpel project
directory
ora:readFile('file:///c:/MyImage.gif') will read file from c:\ directory
ora:readFile('http://www.oracle.com/MyImage.gif')
```

The new BPEL copy statement is as follows:

```
<copy>
  <from expression="string('image/gif')"/>
  <to variable="varNotificationReq" part="EmailPayload" query=
"/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:MimeT
ype"/>
</copy>
<copy>
  <from expression="string('MyImage.gif')"/>
  <to variable="varNotificationReq" part="EmailPayload" query=
"/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:BodyP
artName"/>
</copy>
<copy>
  <from expression="ora:readFile('file:///c:/MyImage.gif')"/>
  <to variable="varNotificationReq" part="EmailPayload" query=
"/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:Conte
ntBody"/>
</copy>
```

See: `SOA_ORACLE_`
`HOME\bpel\samples\tutorials\130.SendEmailWithAttachments`
for an example of sending attachments using e-mail

23.3.1.2 Sending Actionable E-mails

You can send e-mail notifications through the e-mail notification channel.

1. Edit the Outgoing properties to configure the e-mail driver in the `SOA_ORACLE_`
`HOME\j2ee\oc4j_`
`soa\application-deployments\nsdriver-email\oc4j-connectors.xml`
file. These properties are shown in Step 3 on page 28-25.
2. Set the `NotificationMode` property to ALL or EMAIL to enable notifications in
the `SOA_ORACLE_HOME\j2ee\oc4j_`
`soa\applications\soa-infra\configuration\workflow-notification-configuration.xml`
file.

```
NotificationMode="ALL">
```

23.3.1.3 Formatting the Body of an E-mail Message as HTML

You can format the body of an e-mail message as HTML rather than as straight text. To do this, apply an XSLT transform to generate the e-mail 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 e-mail notification assignment should look as follows:

```
<copy>
  <from
    expression="ora:processXSLT('TransformPositionSummary7.xslt',bpws:
    getVariableData('ClientPositionSummary'))" />
  <to variable="varNotificationReq" part="EmailPayload"
    query="/EmailPayload/ns9:Content/ns9:ContentBody" />
</copy>
```

23.3.2 The SMS Notification Channel

When you select **SMS** from the **Component Palette**, the Edit SMS window appears. [Figure 23-4](#) shows the required SMS notification parameters.

Figure 23-4 Edit SMS Window

1. Enter information for each field as described in [Table 23-2](#).

Table 23-2 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 23.3.4, "Setting E-mail Addresses and Telephone Numbers Dynamically" on page 23-11.
Telephone number	The telephone number to which the message is to be delivered. This can be <i>a</i>) a static telephone number entered at the time the message is created, or <i>b</i>) a telephone number looked up using the identity service, or <i>c</i>) a dynamic telephone number from the payload. The XPath Expression Builder can be used to get the dynamic telephone number from the input.

Table 23–2 (Cont.) SMS Notification Parameters

Name	Description
Subject	Subject of the SMS message. This can be free 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	SMS message body. This must be plain text. This can be free text or dynamic text as described for the Subject parameter.

2. Click **OK**.

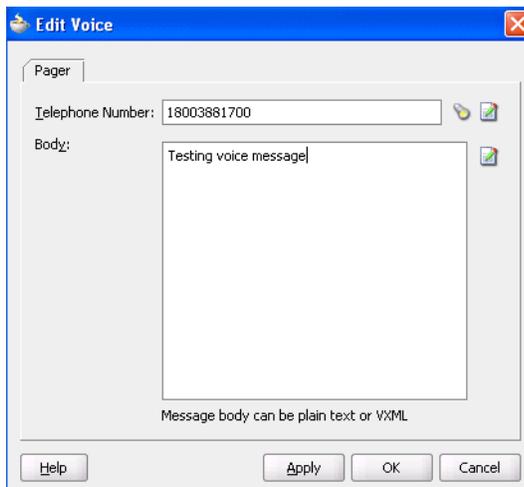
The BPEL fragment that invokes the Oracle User Messaging Service for SMS notification is created.

See Also: *Oracle Fusion Middleware Administrator’s Guide for Oracle SOA Suite* for details about SMS configuration instructions to perform outside of Oracle JDeveloper

23.3.3 The Voice Notification Channel

When you select **Voice** from the **Component Palette**, the Edit Voice window appears. [Figure 23–5](#) shows the required voice notification parameters.

Figure 23–5 Edit Voice Window



1. Enter information for each field as described in [Table 23–3](#).

Table 23–3 Voice Notification Parameters

Name	Description
Telephone number	The telephone number to which the message is to be delivered. This can be <i>a)</i> a static telephone number entered at the time the message is created, or <i>b)</i> a telephone number looked up using the identity service, or <i>c)</i> a dynamic telephone number from the payload. The XPath Expression Builder can be used to get the dynamic telephone number from the input.
Body	Message body. This can be plain text or XML. Also, this can be free 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 Oracle User Messaging Service for voice notification is created.

See Also: *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details about voice configuration instructions to perform outside of Oracle JDeveloper

23.3.4 Setting E-mail Addresses and Telephone Numbers Dynamically

You may need to set e-mail 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 e-mail address or telephone number directly from the payload, use the following XPath:

```
bpws:getVariableData('<variable name>', '<part>', 'input_xpath_to_get_an_address')
```

For example, to get the e-mail 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 e-mail address or telephone number dynamically from the payload, use the following XPath:

```
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 23–4](#) lists the property names that can be used in this XPath function.

Table 23–4 Properties for the Dynamic User XPath Function

Property Name	Description
mail	Look up a user's e-mail 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 e-mail address of the user identified by the variable `inputVariable`, part payload, and query `/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');
```

23.3.5 Selecting Notification Recipients by Browsing the User Directory

You can select users or groups to whom you want to send notifications by browsing the user directory (OID, JAZN/XML, LDAP, and so on) that is configured for use by Oracle BPEL Process Manager. Click the first icon (the flashlight) to the right of **To** (or any recipient field) on any assignee window to start the Identity Lookup dialog.

See Also: [Chapter 28, "Human Task Services"](#) for additional details about using the Identity lookup dialog

23.3.6 Setting Automatic Replies to Unprocessed Messages

The Oracle User Messaging 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 `ResponseMessages.properties` file. This file is located in the `ORACLE_HOME\admin\template\bpel\services\workflow-notification-resource` directory.

```
# 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. Please 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, please 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, please \
    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, please \
    use the response link that was included with your notification.

ResponseNotProcessed=Your response to notification could not be processed. \
    Please login to worklist application for more details.

ResponseProcessed=Your response to notification was successfully processed.
```

You can also internationalize the messages. `ResponseMessages.properties` is a resource for `Java PropertyResourceBundle`. Therefore, you can create one property file for each language that must be supported.

See Also: Sun Microsystem's documentation on `PropertyResourceBundle` for additional details about internationalization

<http://java.sun.com/>

23.3.7 XML Validation Failure with the Oracle User Messaging Service

If you set the `validateXML` property to `true` (the default is `false`) in the `ORACLE_HOME/j2ee/home/applications/soa-infra/configuration/bpel-config.xml` file, each message exchanged with each of the Web services is validated against its schema. However, Oracle User Messaging Service fails during XML validity checks at run time. This is because the BPEL variables exchanged with the Oracle User Messaging Service are not completely initialized in the BPEL process. Part of the initialization happens in the service itself.

23.4 Summary

This chapter describes how you can send an e-mail, voice, IM, or SMS from Oracle BPEL Process Manager.

Oracle BPEL Process Manager Sensors

Using sensors, you can specify BPEL activities, variables, and faults that you want to monitor during run time. This chapter describes how to use and set up sensors for a BPEL process. This chapter also describes how to create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects in an Oracle BAM Server.

This chapter contains the following topics:

- [Section 24.1, "Use Cases for Sensors"](#)
- [Section 24.2, "Introduction to Sensor Concepts"](#)
- [Section 24.3, "Implementing Sensors and Sensor Actions in Oracle JDeveloper"](#)
- [Section 24.4, "Sensors and Oracle Enterprise Manager 11g Application Server Control Console"](#)
- [Section 24.5, "Sensor Integration with Oracle Business Activity Monitoring"](#)
- [Section 24.6, "Sensor Public Views"](#)
- [Section 24.7, "Sensor Actions XSD File"](#)
- [Section 24.8, "Summary"](#)

24.1 Use Cases for Sensors

Using sensors is demonstrated in the sample `125.ReportsSchema`. The sample uses sensors to identify key data during an employee update process and a sensor action to publish information about the update to the database.

Note: The sample demonstrated in `125.ReportsSchema` is not included in the `soa-samples.zip` file for beta 3.

24.2 Introduction to Sensor Concepts

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 suitcase:

- *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 24.3.1, "Configuring Sensors"](#) on page 24-3 and [Section 24.3.2, "Configuring Sensor Actions"](#) on page 24-6 for how these files are created.

After you define sensors for a BPEL process, you must configure sensor actions to publish the data of the sensors to an endpoint. You can publish sensor data to the BPEL reports schema, which is located in the BPEL dehydration store, to a JMS queue or topic, or to a custom Java class.

The following information is required for a sensor action:

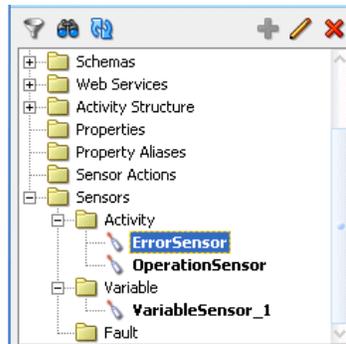
- Name
- Publish type

The publish type specifies the destination where the sensor data must be presented. You can configure the following publish types:

- Database—used to publish the sensor data to the reports schema in the database. The sensor data can then be queried using SQL.
 - JMSQueue—used to publish the sensor data to a JMS queue
 - JMSTopic—used to publish the sensor data to a JMS topic
 - Custom—used to publish 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

24.3 Implementing 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 24-1](#).

Figure 24–1 Sensors and Sensor Actions Displayed in Oracle JDeveloper

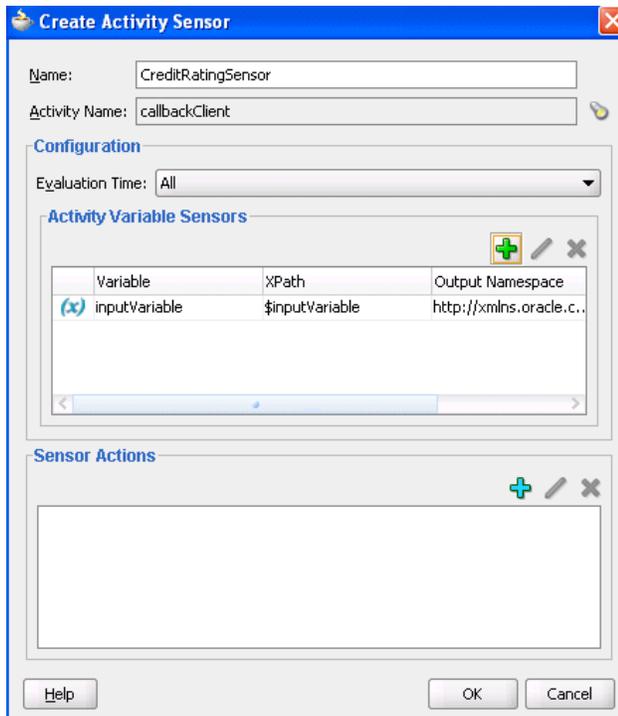
You typically add or edit sensors as part of the BPEL modeling of activities, faults, and variables. You can add sensor actions by right-clicking the **Sensor Actions** folders and selecting **Create > Sensor Action**. To add activity sensors, variable sensors, or fault sensors, expand the **Sensors** folder, right-click the appropriate **Activity**, **Variable**, or **Fault** subfolder, and click **Create**.

Using LoanDemoPlus as an example, the following sections describe how to configure sensors and sensor actions.

See Also: The LoanDemoPlus tutorial, at *SOA_Oracle_Home\bpel\samples\demos*

24.3.1 Configuring Sensors

If you are monitoring the LoanFlow application, you may want to know when the `getCreditRating` scope is initiated, when it is completed, and, at completion, what the credit rating for the customer is. The solution is to create an activity sensor for the `getCreditRating` scope in Oracle JDeveloper, as shown in [Figure 24–2](#). Activities that have sensors associated with them are identified with a magnifying glass in Oracle JDeveloper.

Figure 24–2 Creating an Activity Sensor

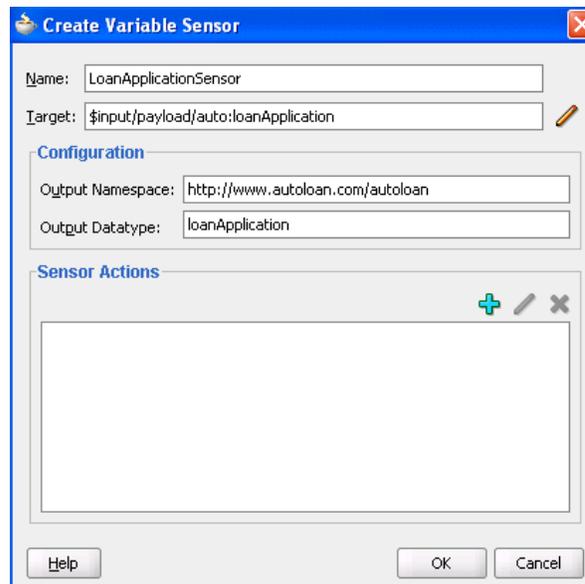
The **Evaluation Time** attribute shown in [Figure 24–2](#) controls the point at which the sensor fires. You can select from the following:

- **All**—Monitoring occurs during all of the preceding phases.
- **Activation**—The sensor fires just before the activity is executed.
- **Completion**—The sensor fires just after the activity is executed.
- **Fault**—The sensor fires if a fault occurs during the execution of the activity. Select this value only for sensors that monitor simple activities.
- **Compensation**—The sensor fires when the associated scope activity is compensated. Select this value only for sensors that monitor scopes.
- **Retry**—The sensor fires when the associated invoke activity is retried.

A new entry is created in the `bpel_process_name_sensor.xml` file, as follows:

```
<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>
```

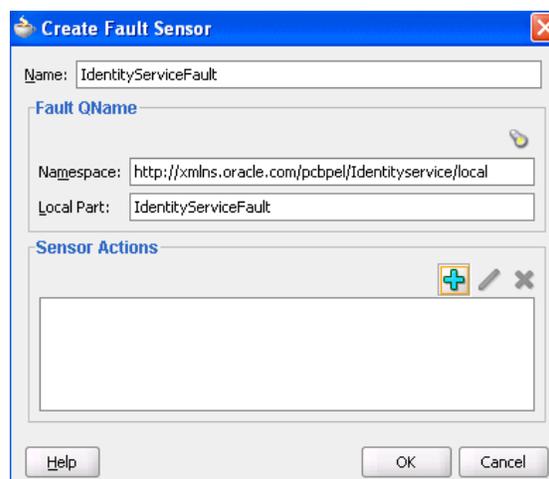
If you want to record all the incoming loan requests, create a variable sensor for the variable `input`, as shown in [Figure 24–3](#).

Figure 24–3 Creating a Variable Sensor

A new entry is created in the `bpel_process_name_sensor.xml` file, as follows:

```
<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 24–4](#).

Figure 24–4 Creating a Fault Sensor

A new entry is created in the `bpel_process_name_sensor.xml` file, as follows:

```
<sensor sensorName="IdentityServiceFault"
  classname="oracle.tip.pc.services.reports.dca.agents.BpelFaultSensorAgent"
  kind="fault">
```

```

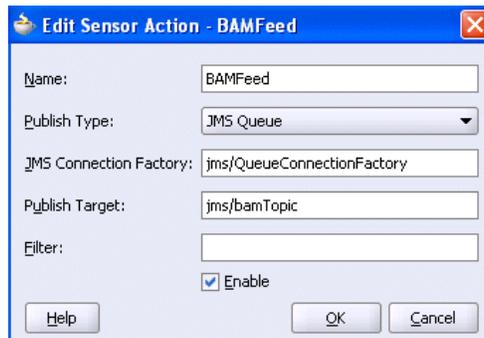
        target="is:identityServiceFault">
    <faultConfig/>
</sensor>

```

24.3.2 Configuring Sensor Actions

When you create sensors, you identify the activities, variables, and faults you want to monitor during run time. 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 24–5](#), and associate it with the `LoanApplicationSensor`.

Figure 24–5 Creating a Sensor Action



A new entry is created in the `bpel_process_name_sensorAction.xml` file, as follows:

```

<action name="BAMFeed"
        enabled="true"
        publishType="JMSQueue"
        publishTarget="jms/bamTopic">
  <sensorName>LoanApplicationSensor</sensorName>
  <property name="JMSConnectionFactory">
    jms/QueueConnectionFactory
  </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 24–6](#), and associate it with both `CreditRatingSensor` and `LoanApplicationSensor`.

Figure 24–6 Creating an Additional Sensor Action



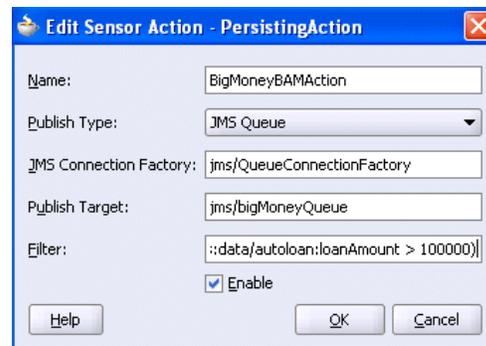
A new entry is created in the `bpel_process_name_sensorAction.xml` file, as follows:

```
<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 24-7](#).

Figure 24-7 *Creating a Sensor Action with a Filter*



A new entry is created in the `bpel_process_name_sensorAction.xml` file, as follows:

```
<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="JMSConnectionFactory">
    jms/QueueConnectionFactory
  </property>
</action>
```

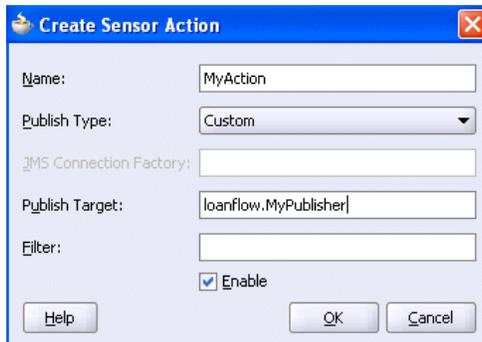
Note:

- 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 24–8. The name in the **Publish Target** field denotes a fully qualified Java class name that must be implemented.

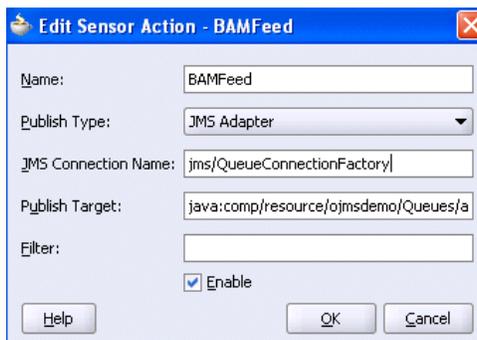
Figure 24–8 Using the Custom Publish Type



24.3.3 Publishing 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 24–9](#).

Figure 24–9 Using the JMS Adapter Publish Type



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

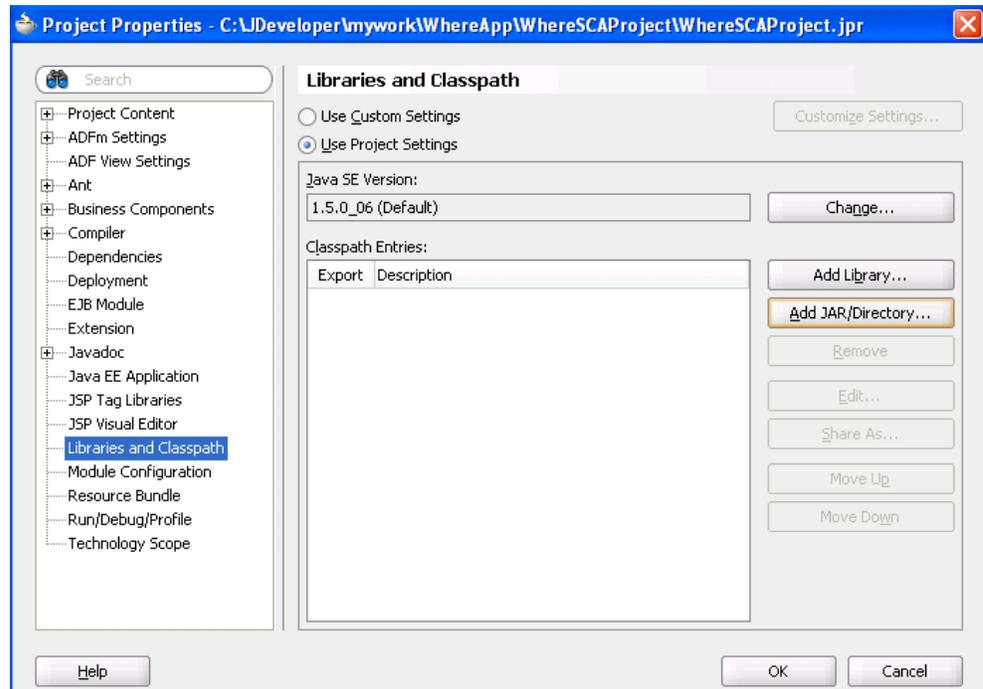
If you select the JMS Adapter publish type, you must create an entry in the `oc4j-ra.xml` file. Use the JMS connection name property in the Sensor Actions dialog to point to the proper entry in the `oc4j-ra.xml` file.

See Also: *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide* for details about the JMS adapter

24.3.4 Creating a Custom Data Publisher

To create a custom data publisher, double-click your BPEL project in Oracle JDeveloper and do the following:

1. Select **Project Properties > Libraries > Add Jar/Directory** from the **Tools** main menu.
2. Browse and select `SOA_ORACLE_HOME\lib\java\shared\oracle.soainfra.common\11.1.1\orabpel.jar`.



3. Create a new Java class.
The package and class name must match the publish target name of the sensor action.
4. 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.
5. Using the Oracle JDeveloper editor, implement the publish method of the `DataPublisher` interface, as shown in the following sample 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());
    }
}

```

6. 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 of the Oracle BPEL Server.

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.

24.3.5 Registering 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 follows:

```

<composite name="JMSQFComposite" 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 the preceding code sample.

24.4 Sensors and Oracle Enterprise Manager 11g Application Server Control Console

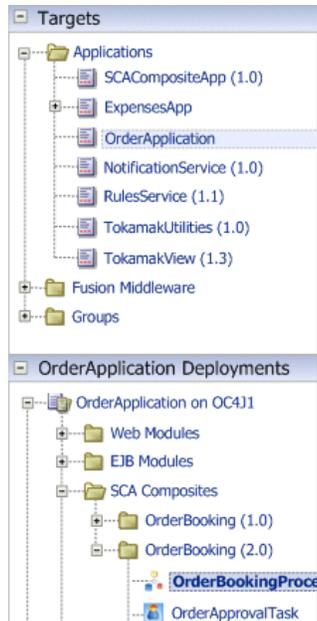
The Oracle Enterprise Manager 11g Application Server Control Console provides support to view the metadata of sensors and sensor actions as well as the sensor data created as part of the process execution.

1. Access the Oracle Enterprise Manager 11g Application Server Control Console at `http://localhost:port/em`
or select **Start > All Programs > Oracle - Oracle_Home > Oracle Application Server Control**.
2. Log in with `fmwadmin/password` when prompted.

24.4.1 Viewing Sensor and Sensor Action Definitions

After the composite application that includes the BPEL process service component is deployed, you can view the definitions of sensors and sensor actions without going back to Oracle JDeveloper.

1. Click the composite application to view.
2. Click the BPEL process service component of the composite application for which you want to see sensor definitions.



3. Click the **Configuration** tab.
4. Click the **Sensors** subtab.
5. Select **Sensor Definitions from the View list**. A page similar to [Figure 24–11](#) displays definitions of the sensors in the BPEL process.

Figure 24–10 Sensor Definitions

The screenshot shows the Oracle Enterprise Manager 11g Application Server Control Console interface for the OrderBookingProcess (BPEL Process). The 'Sensors' tab is selected, and the 'Sensor Definitions' view is active. The page displays three tables of sensor definitions:

Activity Sensors			
Sensor Name	Activity Name	Filter	Evaluation Time
ActivitySensor_Invoke_AssignRating_CustID	assignRating		activation
ActivitySensor_Completion_AssignRating_ContactType	assignRating		completion
ActivitySensor_Completion_receiveSM_SupplierPrice	receiveSM		completion
ActivitySensor_Completion_GetCreditRating_Rating	getCreditRating		completion

Variable Sensors				
Sensor Name	Query	Filter	Namespace	Datatype
VariableSensor_SMPrice	\$receiveSM_onResult_InputVariable/payload/ns2:Supplier...		http://www.w3.org/2001/XMLSchema	decimal
VariableSensor_Rating	\$invokeCR_process_OutputVariable/payload/ns1:rating		http://www.w3.org/2001/XMLSchema	int
VariableSensor_Payload	\$inputVariable/payload/ns2:PurchaseOrder		http://www.globalcompany.com/ns/...	PurchaseOrderType

Fault Sensors		
Sensor Name	Fault	Filter
FaultSensor_NegativeRating	ns1:NegativeCredit	

Sensor Definition Details

Sensor: ActivitySensor_Completion_GetCreditRating_Rating Type: Activity Sensor

Variables		
Query	Namespace	Datatype
\$receiveSM_onResult_InputVariable/payload/ns2:Supplier...	http://www.w3.org/2001/XMLSchema	decimal
\$invokeCR_process_OutputVariable/payload/ns1:rating	http://www.w3.org/2001/XMLSchema	int
\$inputVariable/payload/ns2:PurchaseOrder	http://www.globalcompany.com/ns/...	PurchaseOrderType

Actions	
Action Name	Description
ReportsSensorAction	Reports Schema

6. Select **Action Definitions** from the **View** list. A page similar to [Figure 24–11](#) displays definitions of the sensor actions in the BPEL process.

Figure 24–11 Sensor Action Definitions

The screenshot displays the Oracle Enterprise Manager 11g Application Server Control Console for the OrderBookingProcess (BPEL Process). The interface includes a top navigation bar with tabs for Dashboard, Configuration, Policies, Instances, and Faults and Errors. The Sensors tab is active, showing a table of Sensor Actions and an Action Details section for the ReportsSensorAction.

Sensor Actions

Sensor Actions	Enabled?	Publish Name	Publish Type	Publish Target
ReportsSensorAction	Yes	ReportsSchemaPublisher	BpelReportsSchema	Reports Schema
Another Action	<YES / NO>	<name>	<type>	<target>
Another Action	<YES / NO>	<name>	<type>	<target>
Another Action	<YES / NO>	<name>	<type>	<target>

Action Details

Action: ReportsSensorAction

Sensor Name	Type
ActivitySensor_Completion_GetCreditRating_Rating	Activity
ActivitySensor_Completion_receiveSM_SupplierPrice	Activity
VariableSensor_Payload	Variable
VariableSensor_Rating	Variable
VariableSensor_SMPPrice	Variable
FaultSenser_NegativeRating	Fault

24.4.2 Viewing Sensor Data

You can monitor sensors for which a sensor action is defined.

1. Click the instance number of the BPEL process service component.
2. Click the **Sensors** tab. A page similar to [Figure 24–12](#) is displayed.

This displays the sensor data created as the result of process execution.

Figure 24–12 Sensor Data

Flow Search Results > Flow 123456 > OrderBookingProcess (BPEL Process) :: Instance 212

Flow Interactions **Sensors** Recoverable Faulted Activities

Select a sensor to view its values details below.

Activity Sensors			
Sensor Name	Activity Name	Date	Duration
ActivitySensor_Invoke_AssignRating_CustID	assignRating	May 7, 2006, 11:12:10 AM	Not available
ActivitySensor_Completion_AssignRating_ContactType	assignRating	May 7, 2006, 11:21:15 AM	Not available
ActivitySensor_Completion_receiveSM_SupplierPrice	receiveSM	May 7, 2006, 06:21:15 AM	Not available
ActivitySensor_Completion_GetCreditRating_Rating	getCreditRating	May 7, 2006, 11:21:15 AM	Not available

Variable Sensors	
Sensor Name	Variable Name
VariableSensor_SMPrice	\$receiveSM_onResult_InputVariable/payload/ns2:Supplier...
VariableSensor_Rating	\$invokeCR_process_OutputVariable/payload/ns1:rating
VariableSensor_Payload	\$inputVariable/payload/ns2:PurchaseOrder
VariableSensor_Rating	\$invokeCR_process_OutputVariable/payload/ns1:rating

Fault Sensors			
Sensor Name	Fault	Activity	Message
FaultSensor_NegativeRating	ns1:NegativeCredit	invoke	
FaultSensor_NegativeRating	inputVariable	receivelInput	ns1:PurchaseOrder xmlns:ns1="http://www.g...

Variable Values

Sensor	VariableSensor_Payload	Type	Variable Sensor
Variables			
Time Stamp	Activity Name	Value	
May 7, 2006, 11:21:15 AM	receivelInput	ns1:PurchaseOrder xmlns:ns1="http://www.globalcompany.com/ns/s...	
May 7, 2006, 06:21:15 AM	assignRating	ns1:PurchaseOrder xmlns:ns1="http://www.globalcompany.com/ns/...	
Value			
ns1:PurchaseOrder xmlns:ns1="http://www.globalcompany.com/ns/sales"> 123456789 kadatiya kirit adatiya Bannerghatta Road bangalore karnataka 560075 Indian 9844433014 kirit.adatiya@oracle.com SelectManufacturing 223344 Good credit, Rating = 560-Selected, Select Manufacturing			
May 7, 2006, 10:09:21 AM	selectManufacturing	ns1:PurchaseOrder xmlns:ns1="http://www.globalcompany.com/ns/...	

Note: Only sensors associated with a database sensor action are displayed in Oracle Enterprise Manager 11g Application Server Control Console. Sensors associated with a JMS queue, JMS topic, or custom sensor action are not displayed.

24.5 Sensor Integration with Oracle Business Activity Monitoring

Oracle Business Activity Monitoring (BAM) enables you to monitor business services and processes in an enterprise, correlate key performance indicators (KPIs), and change business processes or take corrective actions if the business environment changes.

Oracle BAM enables you to build real-time operational dashboards and monitoring and alerting applications over the Web. Using this technology, you build interactive, real-time dashboards and proactive alerts to monitor business services and processes.

You can create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects on an Oracle BAM Server. When you create the sensor action, you can select an Oracle BPEL Process Manager variable or activity sensor that you want to monitor and the data object in Oracle BAM Server in which you want to publish the sensor data. Oracle BAM Server publishes the data objects and types information in WSDL files.

This section contains the following topics:

- [Section 24.5.1, "Creating a Connection to Oracle BAM Server"](#)
- [Section 24.5.2, "Creating a Sensor"](#)
- [Section 24.5.3, "Creating a BAM Sensor Action"](#)

These instructions assume you have installed and configured Oracle BAM.

Note: You can also use Oracle BPEL Process Manager 10.1.3.1 sensor actions to publish sensor data as data objects on Oracle BAM Server 11g.

See Also: *Oracle SOA Suite Administrator's Guide*

24.5.1 Creating a Connection to Oracle BAM Server

Notes:

- Do *not* create a BAM Server connection through the **Resource Palette** that displays when you select **View > Resource Palette**.
 - Only one Oracle BAM Server per BPEL project is currently supported.
-

You must create a connection to Oracle BAM Server to browse the available data objects. This connection information is automatically used during deployment.

1. Select **New** from the **File** main menu in Oracle JDeveloper.
The New Gallery dialog opens.
2. Choose **Connections** from the **General** category.
3. Select **BAM Connection** in the **Items** list, 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 following connection information about the Oracle BAM Server host.

Field	Description
BAM Web Host	Enter the name of the host on which the BAM report server and Web server are installed. In most cases, the BAM Web 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 (typically <code>bamadmin</code>).
Password	Enter the password of the user name.
HTTP Port	Enter the port number or accept the default value of 8888 . This is the HTTP port for the BAM Web host.

Field	Description
RMI Port	Enter the port number or accept the default value of 9085 . The RMI port is for the BAM report cache, which is part of the Oracle BAM Server.
Use HTTPS	Select this check box if you want 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**.

24.5.2 Creating a Sensor

You must create one of the following types of sensors prior to creating a BAM sensor action:

- A variable sensor. Since you map the sensor data to Oracle BAM Server data objects, only one variable is allowed for the sensor. If the variable has message parts, then there should be only one message part. This variable must not be defined inline in the WSDL. Only XSD element definitions are supported.
- An activity sensor containing exactly one sensor variable.

See Also: [Section 24.3, "Implementing Sensors and Sensor Actions in Oracle JDeveloper"](#) on page 24-2 for instructions on creating sensors

24.5.3 Creating a BAM Sensor Action

When you create the sensor action, you select the BPEL variable or activity sensor that you want to monitor and the data object in Oracle BAM Server in which you want to publish the sensor data.

1. Go to your BPEL process in Oracle JDeveloper.
2. Right click **Sensor Actions** in the **Structure** window.
3. Select **Create > BAM Sensor Action**.

The Create Sensor Action window appears. You create BAM sensor actions to publish sensor data to data objects on Oracle BAM Server.

4. Enter the following details:

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 24.5.2, "Creating a Sensor" on page 24-17 for mapping sensor data to a data object in Oracle BAM Server.

Field	Description
Data Object	<p>Click the flashlight icon to open the BAM Data Object Chooser window to select the data object in Oracle BAM Server in which you want to publish the sensor data.</p> <p>If you have not already created a connection to Oracle BAM Server in order to select data objects, click the icon in the upper right corner of the BAM Data Object Chooser window.</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 (selected in the Sensor list) and the Oracle BAM Server data object (selected in the Data Object list). You can also invoke a mapper window by clicking the Create Mapping icon (second icon) or Edit Mapping icon (third icon).
Network Protocol	Select RMI , HTTP , or HTTPS as the network protocol to use. Oracle BAM Server publishes the data objects and types information in WSDL files. It uses the selected protocol to transport these files.
Enable Batching	The data cached by default by the Oracle BAM component of the Oracle BPEL Process Manager run time is flushed (sent) to Oracle BAM Server periodically. The Oracle BAM component may decide to send data prior to a batch timeout if the cache has a number of data objects between automatically defined lower and upper limit values. Disable batching by unselecting this check box.

An example of the Create Sensor Action window with a selected data object is shown below.

Create Sensor Action

Action Name:

A BAM sensor action must be associated with a variable sensor or with an activity sensor containing one sensor variable. The variable could either be an XML element or must have exactly one message part. The schema definition of the variable must come from an XSD file ([Inline WSDL schema definitions are not supported](#)). Please select the sensor below:

Sensor:

The BAM sensor action needs to transform the BPEL variable into a data object in the BAM server. Specify the BAM server data object.

Data Object:

BAM Operation and Keys

Operation:

Available Keys:

- _Description
- _Last_Modified

Selected Keys:

- _Salesperson
- _Quantity
- _Total

Map File:

Clicking 'Create Mapping' or 'Edit Mapping' will save the sensor action, close this dialog and open the mapper in IDE main window.

Network protocol: Enable Batching

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 window, you must explicitly save the BPEL file.

5. Click **OK** to close the Create Sensor Action window.

24.6 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 11g Application Server Control Console.

A set of public views is exposed to allow SQL access to sensor values from literally any application interested in the data.

24.6.1 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, OracleAS Portal, and so on) to query the sensor values using SQL.

Note: In Table 24–1 through Table 24–4, 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.

BPEL_PROCESS_INSTANCES

This view provides an overview of all the process instances of Oracle BPEL Process Manager.

Table 24–1 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	100	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	100	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	100	--	N	User-defined label
COMPONENT_NAME	VARCHAR2	100	--	N	User-defined component name
TITLE	VARCHAR2	50	--	Y	User-defined title of the BPEL process
STATE	NUMBER	--	--	Y	State of the BPEL process instance
STATE_TEXT	VARCHAR2	--	--	Y	Text presentation of the state attribute
PRIORITY	NUMBER	--	--	Y	User-defined priority of the BPEL process instance
STATUS	VARCHAR2	100	--	Y	User-defined status of the BPEL process
STAGE	VARCHAR2	100	--	Y	User-defined stage property of a BPEL process
CONVERSATION_ID	VARCHAR2	100	--	Y	User-defined conversation ID of a BPEL process
CREATION_DATE	TIMESTAMP	--	--	N	Creation time stamp of the process instance
MODIFY_DATE	TIMESTAMP	--	--	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

BPEL_ACTIVITY_SENSOR_VALUES

This view contains all the activity sensor values of the monitored BPEL processes.

Table 24–2 BPEL_ACTIVITY_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	ID of process instance
APPLICATION_NAME	VARCHAR2	100	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	100	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	100	--	N	User-defined label
COMPONENT_NAME	VARCHAR2	100	--	N	User-defined component name
SENSOR_NAME	NVARCHAR2	100	U1,2	N	The name of the sensor that fired
SENSOR_TARGET	NVARCHAR2	256	--	N	The target of the fired sensor
ACTION_NAME	NVARCHAR2	100	U1,3	N	The name of the sensor action
ACTION_FILTER	NVARCHAR2	256	--	Y	The filter of the action
CREATION_DATE	TIMESTAMP	--	--	N	The creation date of the activity sensor value
MODIFY_DATE	TIMESTAMP	--	--	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, Y, or N
ACTIVITY_NAME	NVARCHAR2	100	--	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	20	--	N	The evaluation point of the activity sensor
ERROR_MESSAGE	NVARCHAR2	2000	--	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

BPEL_FAULT_SENSOR_VALUES

This view contains all the fault sensor values.

Table 24–3 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	100	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	100	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	100	--	N	User-defined label
COMPONENT_NAME	VARCHAR2	100	--	N	User-defined component name
SENSOR_NAME	NVARCHAR2	100	U1,2	N	The name of the sensor that fired
SENSOR_TARGET	NVARCHAR2	256	--	N	The target of the fired sensor
ACTION_NAME	NVARCHAR2	100	U1,3	N	The name of the sensor action
ACTION_FILTER	NVARCHAR2	256	--	Y	The filter of the action
CREATION_DATE	TIMESTAMP	--	--	N	The creation date of the activity sensor value
MODIFY_DATE	TIMESTAMP	--	--	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	100	--	N	The name of the BPEL activity
ACTIVITY_TYPE	VARCHAR2	30	--	N	The type of the BPEL activity
MESSAGE	CLOB	--	--	Y	The fault message

BPEL_VARIABLE_SENSOR_VALUES

This view contains all the variable sensor values.

Table 24–4 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	100	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	100	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	100	--	N	User-defined label

Table 24–4 (Cont.) BPEL_VARIABLE_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
COMPONENT_NAME	VARCHAR2	100	--	N	User-defined component name
SENSOR_NAME	NVARCHAR2	100	U1,2	N	Name of the sensor that fired
SENSOR_TARGET	NVARCHAR2	256	--	N	Target of the sensor
ACTION_NAME	NVARCHAR2	100	U1,3	N	Name of the action
ACTION_FILTER	NVARCHAR2	256	--	Y	Filter of the action
ACTIVITY_SENSOR	NUMBER	--	--	Y	ID of corresponding activity sensor value
CREATION_DATE	TIMESTAMP	--	--	N	Creation date
TS_DATE	DATE	--	--	N	Date portion of <code>creation_date</code>
TS_HOUR	NUMBER	--	--	N	Hour portion of <code>creation_date</code>
VARIABLE_NAME	NVARCHAR2	256	--	N	The name of the BPEL variable
EVAL_POINT	VARCHAR2	30	--	Y	Evaluation point of the corresponding activity sensor
CRITERIA_SATISFIED	VARCHAR2	1	--	Y	NULL, Y, or N
TARGET	NVARCHAR2	256	--	--	--
UPDATER_NAME	NVARCHAR2	100	--	N	The name of the activity or event that updated the variable
UPDATER_TYPE	NVARCHAR2	100	--	N	The type of the BPEL activity or event
SCHEMA_NAMESPACE	NVARCHAR2	256	--	Y	Namespace of variable sensor value
SCHEMA_DATATYPE	NVARCHAR2	256	--	Y	Datatype of the variable sensor value
VALUE_TYPE	SMALLINT	--	--	N	The value type of the variable (corresponds to <code>java.sql.Types</code> values)
VARCHAR2_VALUE	NVARCHAR2	2000	--	Y	The value of string-like variables

24.7 Sensor Actions XSD File

The section provides a sample sensor action schema that you can import into Oracle JDeveloper. This schema is also relevant to custom data publishers.

```
<?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: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: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: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>

<!--
    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"
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>
```

24.8 Summary

This chapter describes how to set up and use sensors to monitor BPEL activities, variables, and faults during run time. This chapter also describes how to create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects in an Oracle BAM Server.

Business Rule Service Component

This chapter describes how to use a business rule service component to integrate a SOA composite application with a business rules engine.

This chapter contains the following topics:

- [Section 25.1, "Business Rule Concepts"](#)
- [Section 25.2, "Business Rule Architecture"](#)
- [Section 25.3, "Use Cases for Integration of Business Processes and Business Rules"](#)
- [Section 25.4, "Integration of BPEL Processes, Human Tasks, and Business Rules"](#)
- [Section 25.5, "Deploying a Business Rule"](#)
- [Section 25.6, "Running a Business Rule in a SOA Composite Application"](#)

See Also: Business rule sample files located in the `soa-samples.zip` file

25.1 Business Rule Concepts

This section provides an overview of Oracle SOA Suite support for business rules and business rule engines.

This section contains the following topics:

- [Section 25.1.1, "Business Rules and Business Rule Engines"](#)
- [Section 25.1.2, "Business Rule Service Component"](#)
- [Section 25.1.3, "Oracle SOA Suite"](#)

25.1.1 Business Rules and Business Rule Engines

Business rules are statements that describe the policies of a company. Examples of business rules can include the following:

- All customers that buy more than \$100 worth of products at one time or who are over the age of 65 receive a 10% discount.
- A sales department is notified when inventory quantity is lower than ten and there are more than five pending orders on a given day.
- If the annual income of a customer is less than \$10,000, a loan request is denied.
- If a customer submitted a late payment for a previous purchase, an additional charge of 2% is added to their next purchase.

A business rules engine is a system that manages and executes business rules. A business rule system typically consists of a rule repository, rule author, and rules engine. The rule author allows business rules to be specified separately from application code. Separating the business rules from code allows business analysts to change business policies quickly with graphical tools. The rules engine evaluates the business rule and returns decisions or facts that are then used in the business process. The rules are typically stored in a rule repository in a database or file system.

25.1.2 Business Rule Service Component

Oracle SOA Suite provides support for a business rule service component. A business rule service component is a mechanism for publishing rules and rule sets as a reusable service that can be invoked from multiple business processes. The business rule service component consists of the following components:

- A Web service that wraps the rule session to the underlying rule engine. This 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 has to support both stateless and stateful interactions.
- Rules that are evaluated by the business rule service component using the rule engine. These are defined using the rule author and loaded into the rule repository.
- Metadata that describes facts required for specific rules to be evaluated. Each rule exposed as a service uses different types of facts. These facts must be exposed through XSD definitions. Appropriate WSDL operations must be exposed for rule evaluation.

For example, a credit rating rule set may expect a customer's SSN, and previous loan history as facts, but a pension payment rule may expect an employee's years of service, salary, and age as facts.

The business rule service component can be integrated into a SOA composite application in the following ways:

- Create a business rule service component as a standalone component in the SOA Composite Editor
- Create a business rule service component in the SOA Composite Editor that you later associate with a BPEL process
- Create a business rule service component within the Human Task editor of a human task component

See Also: [Section 25.2, "Business Rule Architecture"](#) on page 25-3

25.1.3 Oracle SOA Suite

Oracle SOA Suite provides support for Oracle Business Rules. Business analysts use Oracle Business Rules to create and change business rules that are separate from the application code. This enables analysts to change business rules without stopping business processes or having to involve programmers.

In Oracle Business Rules, facts are data objects asserted in the rules engine. For example:

- For a car rental company to create a rule to match a driver's age, the driver's age represents the fact used in the rule.

- For a loan company to create a rule denying a loan request to customers whose incomes fall below a specific level, the income level represents the fact used in the rule.

Each data object instance corresponds to a single fact. Rules are expressions that can be evaluated on this factual information.

If an object is re-asserted (whether or not it has changed), the rules engine is updated to reflect the new state of the object. Re-asserting the object does not create a new fact. In order to have multiple facts of a particular fact type, separate object instances must be asserted.

Using the Oracle Business Rules Designer (Rule Designer) in Oracle JDeveloper, you create rules that operate on facts that are part of a data model. You make business objects and their methods known to Oracle Business Rules using fact definitions.

Oracle Business Rules consist of the following key components:

- Rule Designer — An editor integrated with Oracle JDeveloper that enables you to create and edit rules.
- Oracle Business Rules Rules Engine (Rules Engine) — A Java library that applies rules to facts and defines and processes rules. The Rules Engine defines a declarative rule language, provides a language processing engine, and provides debugging tools.

Oracle SOA Suite provides support for a Rules Engine file repository to store business rules. A repository stores dictionaries. A dictionary is a set of XML files that stores the application's rule sets and the data model. Rule sets are a group of business rules that are typically evaluated together and generated as one unit. Dictionaries can have different versions. Dictionaries and dictionary versions can be created, deleted, exported, and imported into a repository.

- Oracle Business Rules SDK (Rules SDK) — A Java library that provides business rule management features to use for writing customized rules programs.
- Oracle Business Rules RL Language (RL Language) — A language that defines the syntax for Rules Engine programs. RL Language includes an intuitive Java-like syntax for defining rules that supports Java semantics.

This integration provides the following benefits:

- Dynamic processing (provides for intelligent routing, validation of policies within a process, and constraint checks)
- Integration with adhoc human tasks (provides policy-based task assignment, various escalation policies, and load balancing of tasks)
- Integration with business activity monitoring (sends alerts based on certain policies and dynamic processing-based key performance indicator (KPI) reasoning)

See Also: *Oracle Fusion Middleware User's Guide for Oracle Business Rules*

25.2 Business Rule Architecture

This section describes the business rule architecture.

This section contains the following topics:

- [Section 25.2.1, "Business Rule Service Component Metadata File"](#)

- [Section 25.2.2, "Rule Dictionary"](#)
- [Section 25.2.3, "SCA Component Type"](#)
- [Section 25.2.4, "Decision Services"](#)

25.2.1 Business Rule Service Component Metadata File

The business rule metadata file (*business_rule_name.decs*) defines the contract between the components involved in the interaction of the business rule with the design time and backend Rules Engine.

The following example shows a *business_rule_name.decs* file for a business rule service component named `AutoLoanRules`:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<decisionServices xmlns="http://xmlns.oracle.com/bpel/rules" name="AutoLoanRules">
  <ruleEngineProvider name="OracleRulesSDK" provider="Oracle_11.0.0.0">
    <repository type="SCA-Archive">
      <path>AutoLoanComposite/oracle/rules/AutoLoanRules.rules</path>
    </repository>
  </ruleEngineProvider>
  <decisionService targetNamespace="http://xmlns.oracle.com/creditrating/Rating"
    ruleEngineProviderReference="OracleRulesSDK" name="CreditRatingService">
    <catalog>AutoLoanRules</catalog>
    <ruleset>CreditRatingRuleset</ruleset>
    <pattern name="Assert">
      <arguments>
        <assert>creditrating.Ratingrequest</assert>
      </arguments>
    </pattern>
    <pattern name="AssertExecute">
      <arguments>
        <assert>creditrating.Ratingrequest</assert>
      </arguments>
    </pattern>
    <pattern name="Watch">
      <arguments>
        <watch>creditrating.Rating</watch>
      </arguments>
    </pattern>
    <pattern name="AssertExecuteWatchStateless">
      <arguments>
        <assert>creditrating.Ratingrequest</assert>
        <watch>creditrating.Rating</watch>
      </arguments>
    </pattern>
    <pattern name="AssertExecuteWatchStateful">
      <arguments>
        <assert>creditrating.Ratingrequest</assert>
        <watch>creditrating.Rating</watch>
      </arguments>
    </pattern>
  </decisionService>
  <decisionService targetNamespace="http://xmlns.oracle.com/loanoffer/Advisor"
    ruleEngineProviderReference="OracleRulesSDK" name="LoanAdvisorService">
    <catalog>AutoLoanRules</catalog>
    <pattern name="CallFunctionStateless">
      <arguments>
        <call>AutoLoanRules.computeLoanAdvise</call>
      </arguments>
    </pattern>
  </decisionService>
</decisionServices>
```

```

    </pattern>
    <pattern name="CallFunctionStateful">
      <arguments>
        <call>AutoLoanRules.computeLoanAdvise</call>
      </arguments>
    </pattern>
  </decisionService>
</decisionServices>

```

The configuration file includes the following elements:

- `ruleEngineProvider` — Specifies metadata about the backend Rules Engine connection. Apart from the Rules Engine provider, information on the rule repository is specified.
- `decisionService` — Consists of a name, an optional target namespace, and a mandatory reference to the Rules Engine to use for the interaction. A complete interaction with the Rules Engine is specified, which includes the catalog and catalog version to use, and the rule set to execute. Various interaction patterns are supported. Apart from the name of the pattern, the pattern signature is specified in terms of input and output facts or function names.

25.2.2 Rule Dictionary

The `business_rule_name.decs` file includes details about the rule dictionary to use:

```

<ruleEngineProvider name="OracleRulesSDK" provider="Oracle_11.0.0.0.0">
  <repository type="SCA-Archive">
    <path>AutoLoanComposite/oracle/rules/AutoLoanRules.rules</path>
  </repository>
</ruleEngineProvider>

```

The repository type is set to `SCA-Archive` for business rule service components. This indicates that the rule dictionary is located in the service component architecture archive. The path is relative and interpreted differently by the following:

- **Design time** — The path is prefixed with `Oramds : /`. Metadata service (MDS) APIs open the rule dictionary. Therefore, the full path to the dictionary is as follows:

```
Oramds:/AutoLoanComposite/oracle/rules/AutoLoanRules.rules
```

- **Runtime (business rule service engine)** — The business rule service engine uses the metadata manager to open the rule dictionary. The composite name prefix (`AutoLoancomposite`) is removed from the path and the metadata manager assumes the existence of `oracle/rules/AutoLoanRules.rules` relative to the composite home directory.

Notes:

- The concept of dictionary links is not exposed through business rule metadata. It is assumed that all linked dictionaries exist at an appropriate location in the SOA archive.
 - The Web Distributed Authoring and Versioning (WebDAV) rule repository is not supported in release 11g.
-
-

25.2.3 SCA Component Type

An SCA `business_rule_name.componentType` file is included with each business rule service component. This file lists the services exposed by the business rules service component. In the following sample, two services are exposed: `CreditRatingService` and `LoanAdvisorService`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version 1.0 at [5/24/07 9:27 AM]. -->
<componentType xmlns="http://xmlns.oracle.com/sca/1.0">
  <service name="CreditRatingService">
    <interface.wSDL
interface="http://xmlns.oracle.com/creditrating/Rating#wsdl.interface(IDecisionService)"/>
    </service>
  <service name="LoanAdvisorService">
    <interface.wSDL
interface="http://xmlns.oracle.com/loanoffer/Advisor#wsdl.interface(IDecisionService)"/>
    </service>
</componentType>
```

25.2.4 Decision Services

A decision service is a Web service (or SOA) enabler of business rules. It is a service in the sense of a Web service, thus opening the world of business rules to service-oriented architectures (SOA). In release 11g, a decision service consists of metadata and a WSDL contract for the service.

A decision service includes the following elements:

- Target namespace
- Reference to the backend Rules Engine (this is the link to the rule dictionary). Note that `OracleRulesSDK` is the reference name that matches the name of the Rules Engine provider in [Section 25.2.2, "Rule Dictionary"](#) on page 25-5.
- Name (`CreditRatingService` in the following example)
- Additional information about the dictionary name and rule set to use
- List of supported operations (patterns)

The following example shows the most commonly-used interaction with a rules engine, `execute ruleset`.

```
. . .
. . .
<decisionService targetNamespace="http://xmlns.oracle.com/creditrating/Rating"
  ruleEngineProviderReference="OracleRulesSDK" name="CreditRatingService">
  <catalog>AutoLoanRules</catalog>
  <ruleset>CreditRatingRuleset</ruleset>
  <pattern name="Assert">
    <arguments>
      <assert>creditrating.Ratingrequest</assert>
    </arguments>
  </pattern>
  <pattern name="AssertExecute">
    <arguments>
      <assert>creditrating.Ratingrequest</assert>
    </arguments>
  </pattern>
  <pattern name="Watch">
```

```

        <arguments>
            <watch>creditrating.Rating</watch>
        </arguments>
    </pattern>
    <pattern name="AssertExecuteWatchStateless">
        <arguments>
            <assert>creditrating.Ratingrequest</assert>
            <watch>creditrating.Rating</watch>
        </arguments>
    </pattern>
    <pattern name="AssertExecuteWatchStateful">
        <arguments>
            <assert>creditrating.Ratingrequest</assert>
            <watch>creditrating.Rating</watch>
        </arguments>
    </pattern>
</decisionService>
<decisionService targetNamespace="http://xmlns.oracle.com/loanoffer/Advisor"
ruleEngineProviderReference="OracleRulesSDK" name="LoanAdvisorService">
    <catalog>AutoLoanRules</catalog>
    <pattern name="CallFunctionStateless">
        <arguments>
            <call>AutoLoanRules.computeLoanAdvise</call>
        </arguments>
    </pattern>
    <pattern name="CallFunctionStateful">
        <arguments>
            <call>AutoLoanRules.computeLoanAdvise</call>
        </arguments>
    </pattern>
</decisionService>
</decisionServices>

```

Apart from the operations (patterns), the parameter types (or fact types) of operations are specified (and validated later at runtime). Therefore, every decision service exposes a strongly-typed contract.

There is a one-to-one relationship between decision service metadata and the WSDL file being generated for a decision service. Operation `AssertExecuteWatchStateless` provides an example:

```

<pattern name="AssertExecuteWatchStateless">
    <arguments>
        <assert>creditrating.Ratingrequest</assert>
        <watch>creditrating.Rating</watch>
    </arguments>
</pattern>

```

The following WSDL artifacts are generated for `AssertExecuteWatchStateless`:

```

<element name="assertExecuteWatchStateless">
    <complexType>
        <sequence>
            <element name="configURL" type="string" minOccurs="1" maxOccurs="1"/>
            <element name="bpelInstance" type="bpelpm:tBpelProcess" minOccurs="0"
                maxOccurs="1"/>
            <element name="assertList" minOccurs="1" maxOccurs="1">
                <complexType>
                    <sequence>
                        <element ref="ns1:ratingrequest"/>
                    </sequence>
                </complexType>
            </element>
        </sequence>
    </complexType>
</element>

```

```

        </complexType>
      </element>
    </sequence>
    <attribute name="name" type="NCName" use="required"/>
  </complexType>
</element>

```

The preceding syntax also showed the XML-schema definition for the input message of pattern `AssertExecuteWatchStateless`. As an analogy, an XML-schema element is being created for the output message:

```

<element name="assertExecuteWatchStatelessDecision">
  <complexType>
    <sequence>
      <element name="resultList" minOccurs="1" maxOccurs="1">
        <complexType>
          <sequence>
            <element ref="ns1:rating"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
. . .
. . .
<message name="assertExecuteWatchStatelessMessage">
  <part name="payload" element="tns:assertExecuteWatchStateless"/>
</message>
<message name="assertExecuteWatchStatelessDecisionMessage">
  <part name="payload" element="tns:assertExecuteWatchStatelessDecision"/>
</message>
. . .
. . .
<portType name="IDecisionService">
  <operation name="assert">
    <input name="assertInput" message="tns:assertMessage"/>
  </operation>
  <operation name="assertExecute">
    <input name="assertExecuteInput" message="tns:assertExecuteMessage"/>
  </operation>
  <operation name="watch">
    <input name="watchInput" message="tns:watchMessage"/>
    <output name="watchOutput" message="tns:watchDecisionMessage"/>
    <fault name="operationErroredFault" message="tns:decisionServiceError"/>
  </operation>
  <operation name="assertExecuteWatchStateless">
    <input name="assertExecuteWatchStatelessInput"
      message="tns:assertExecuteWatchStatelessMessage"/>
    <output name="assertExecuteWatchStatelessOutput"
      message="tns:assertExecuteWatchStatelessDecisionMessage"/>
    <fault name="operationErroredFault"
      message="tns:decisionServiceError"/>
  </operation>
  <operation name="assertExecuteWatchStateful">
    <input name="assertExecuteWatchStatefulInput"
      message="tns:assertExecuteWatchStatefulMessage"/>
    <output name="assertExecuteWatchStatefulOutput"
      message="tns:assertExecuteWatchStatefulDecisionMessage"/>
    <fault name="operationErroredFault"

```

```

        message="tns:decisionServiceError"/>
    </operation>
</portType>

```

Note the following details about decision services:

- Multiple decision services of a business rule service component *cannot* share an XML schema namespace. This is because there cannot be two messages with the same namespace, but using different XML schema definitions.
- The WSDL contract is relaxed in that at runtime, the input message elements are optional. In the preceding sample, you can pass a `ratingrequest` element, or leave the element empty. The decision service checks for empty parameter elements and ignores them when asserting facts.
- For the Rules Engine, the decision service provides supplemental rule sets. These rule sets are pushed on the rule set stack, but not executed at runtime.
- One decision service can only expose *one* function. This is no real limitation since you may model one decision service per function. The benefit is that you comply with the strongly-typed interface.
- The rule dictionary unifies a decision component, although this is a limitation of design-time and not the decision service.

25.2.4.1 Decision Services that Expose an RL Function

As described in [Section 25.2.4, "Decision Services"](#) on page 25-6, executing a rule set is the most common interaction pattern used with a business rules engine. However, Oracle Business Rules provides a function (with parameters and return type). Functions are useful for setup purposes or to hide the complexity of executing multiple rule sets. The biggest advantage to using a function is the well-defined contract in terms of function parameters and return type.

The decision service has the capability to expose RL functions as services. As with the `execute ruleset` interaction, a decision service that exposes a function is defined by metadata.

```

<decisionService targetNamespace="http://xmlns.oracle.com/RatingSetupService"
    ruleEngineProviderReference="OracleRulesSDK"
    name="RatingSetupService">
    <catalog>CreditRatingDictionary</catalog>
    <pattern name="CallFunctionStateless">
    <arguments>
    <call>cragency.CreditRatingDictionary.setupCreditRatingAgency</call>
    </arguments></pattern>
    <pattern name="CallFunctionStateful">
    <arguments>
    <call>cragency.CreditRatingDictionary.setupCreditRatingAgency</call>
    </arguments>
    </pattern>
</decisionService>

```

For a decision service exposing an RL function, you only need the information about the rule dictionary and the fully-qualified function name. The function signature can be queried from the rule dictionary. Note also that no rule set name must be specified because functions hang off a dictionary.

Note the following details about decision services that expose an RL function:

- As with the `execute ruleset` interaction, a decision service that exposes an RL function is identified by its name, target namespace, reference to a rule dictionary, the dictionary name itself, and the fully-qualified function name.
- Stateless and stateful interactions can be modeled. A stateful interaction is useful when the rule session requires setup (for example, you must set the value of rule variables) prior to executing a rule set or if some status information must be retrieved between rule set executions.

The function signature is needed when the WSDL for the service is created:

```
<element name="callFunctionStateless">
  <complexType>
    <sequence>
      <element name="parameterList" minOccurs="1"maxOccurs="1">
        <complexType>
          <sequence>
            <element ref="ns1:setupParameter"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
    <attribute name="name" type="NCName" use="required"/>
  </complexType>
</element>
<element name="callFunctionStatelessDecision">
  <complexType>
    <sequence>
      <element name="resultList" minOccurs="1" maxOccurs="1">
        <complexType>
          <sequence>
            <element ref="rules:int"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
```

In the preceding sample, the function consists of one parameter of XML schema element `ns1:setupParameter` and a return type of `rules:int`.

The following messages and operations are also part of the WSDL contract:

```
<message name="callFunctionStatelessMessage">
  <part name="payload" element="tns:callFunctionStateless"/>
</message>
<message name="callFunctionStatelessDecisionMessage">
  <part name="payload" element="tns:callFunctionStatelessDecision"/>
</message>

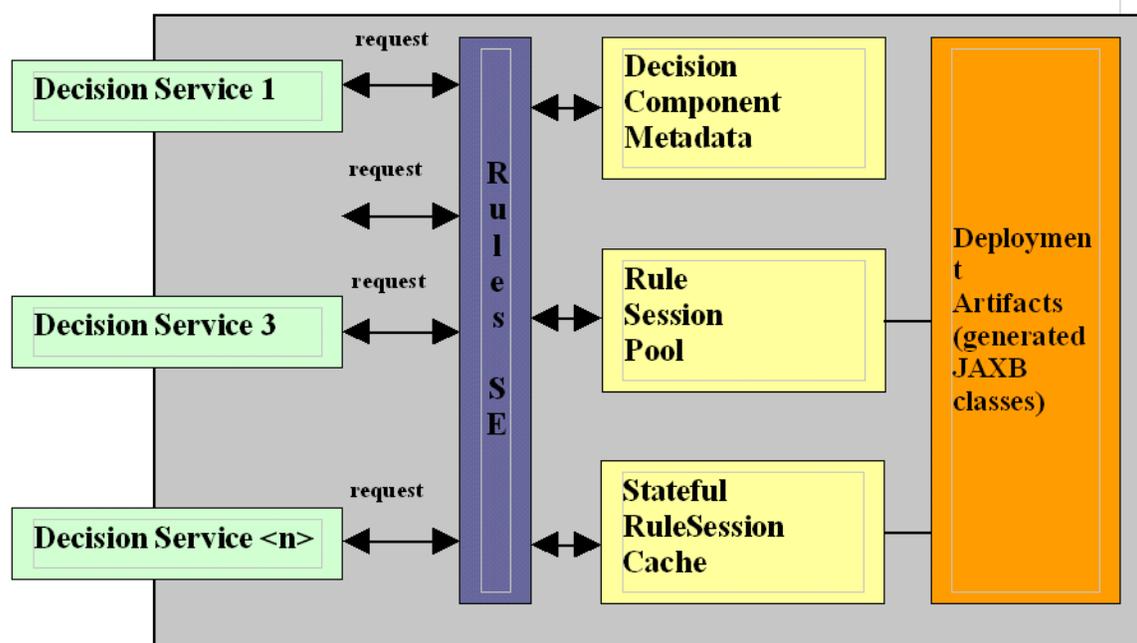
<portType name="IDecisionService">
  <operation name="callFunctionStateless">
    <input name="callFunctionStatelessInput"
      message="tns:callFunctionStatelessMessage"/>
    <output name="callFunctionStatelessOutput"
      message="tns:callFunctionStatelessDecisionMessage"/>
    <fault name="operationErroredFault"
      message="tns:decisionServiceError"/>
  </operation>
</portType>
```

A decision service is a lightweight entity. It consists only of the service description. All other artifacts are shared within a decision component. The heart of runtime is the decision service cache, which is organized in a tree structure. Every decision component owns a subtree of that cache (depending on the composite distinguished name (DN), component, and service name). In this regard, decision services of a decision component share the following data:

- Metadata of the decision component
 - Fact type metadata
 - Function metadata
 - Rule Set metadata
- Rule session pool
 - One rule session pool is created per decision component
 - The rule sessions in the pool are preinitialized with the data model RL and the rule set RL already executed
 - New rule sessions are created on demand
 - Rule sessions can be reused for a configurable number of times
 - The initial size of the rule session pool is configurable
- Stateful rule session cache
 - A special cache is maintained for stateful rule sessions (see [Section 25.2.5, "Stateful Interactions with a Decision Component"](#) on page 25-12)
- Deployment artifacts
 - Decision component deployment can end up in class generation for JAXB fact types. The classes can be shared across the composite.

[Figure 25-1](#) provides an overview.

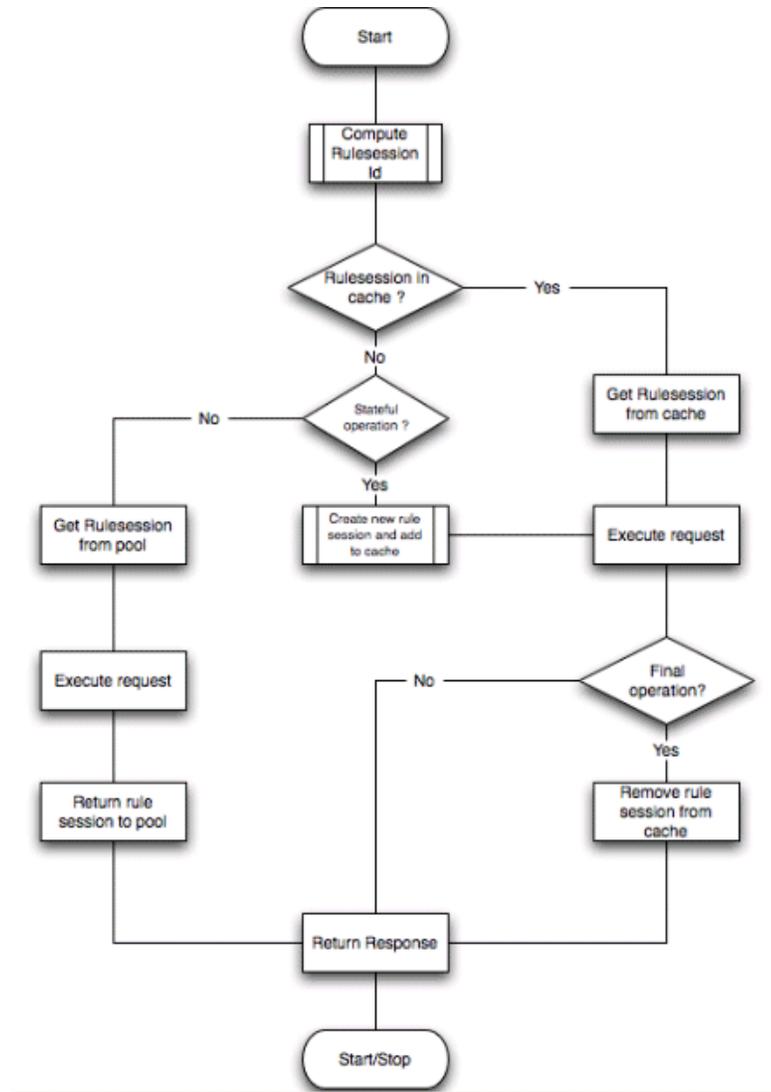
Figure 25-1 Shared Component Data



25.2.5 Stateful Interactions with a Decision Component

Figure 25–2 shows stateful rule session usage for a decision service request.

Figure 25–2 Rule Session Usage for a Decision Service Request



Note the following details about stateful interactions with a decision component:

- The following operations are final. A rule session is removed from the cache after a final operation finishes (therefore, querying for results clears the state).
 - Retrieve the result
 - Assert facts, execute the rule set, and retrieve the result
 - Call the function (stateless)
- The following operations are stateful operations
 - Assert the facts
 - Assert the facts and execute the rule set
 - Call the function (stateful)

- Rule sessions from the cache and those from the pool are mutually exclusive
 - The rule session pool is for simple, stateless interactions only
 - The rule session cache keeps the state of a rule session across decision service requests

25.3 Use Cases for Integration of Business Processes and Business Rules

You can create a 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 rule 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 the loan application to the StarLoan loan agency only. If the customer can wait longer, then route the request to three different loan agencies.
- Externalize decision points in the process — There are typically many conditions that must be evaluated as part of a business process. However, the parameters to these 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 — 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.

See Also: [Section 26.6.5.12, "Advanced Task Routing Using Business Rules"](#) on page 26-42 to create business rules in the Human Task editor of a human task component

25.4 Integration of BPEL Processes, Human Tasks, and Business Rules

Oracle BPEL Process Manager provides the following design-time components that enable you to integrate BPEL process, business rule, and human task service components.

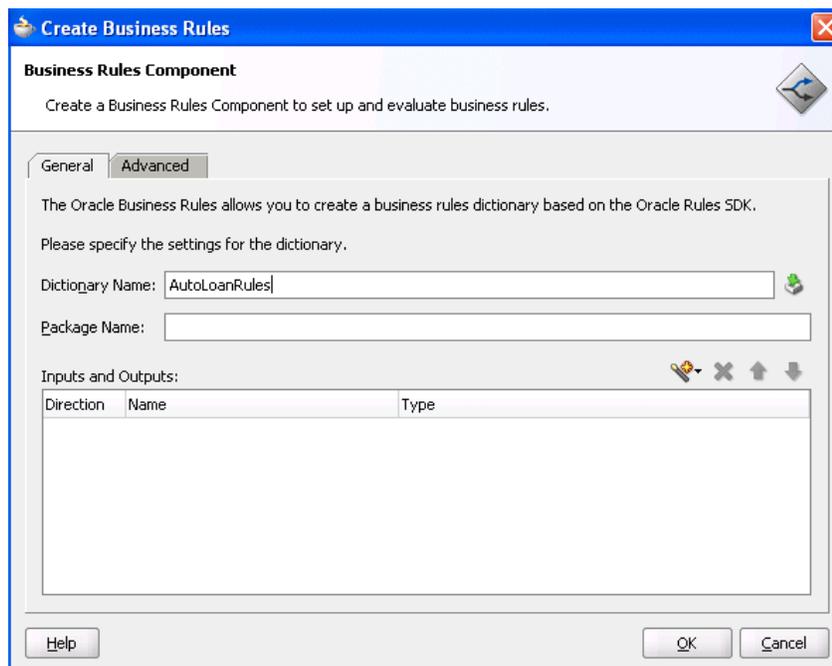
- [Section 25.4.1, "Business Rule Service Component"](#)
- [Section 25.4.2, "Business Rule Activity of a Business Process"](#)
- [Section 25.4.3, "Human Task Component"](#)

See Also: Business rule sample files and tutorial that uses all three service components located in the `soa-samples.zip` file

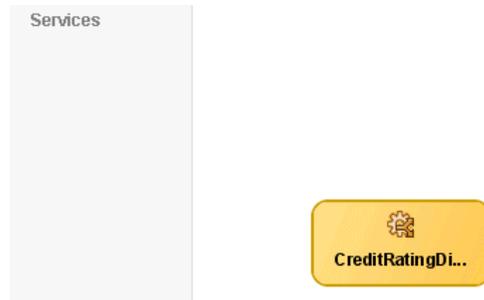
25.4.1 Business Rule Service Component

The business rule service component enables you to integrate your SOA composite application with a business rule (for example, a rule set or function). This enables you to execute the business rule and make business decisions based on the rules.

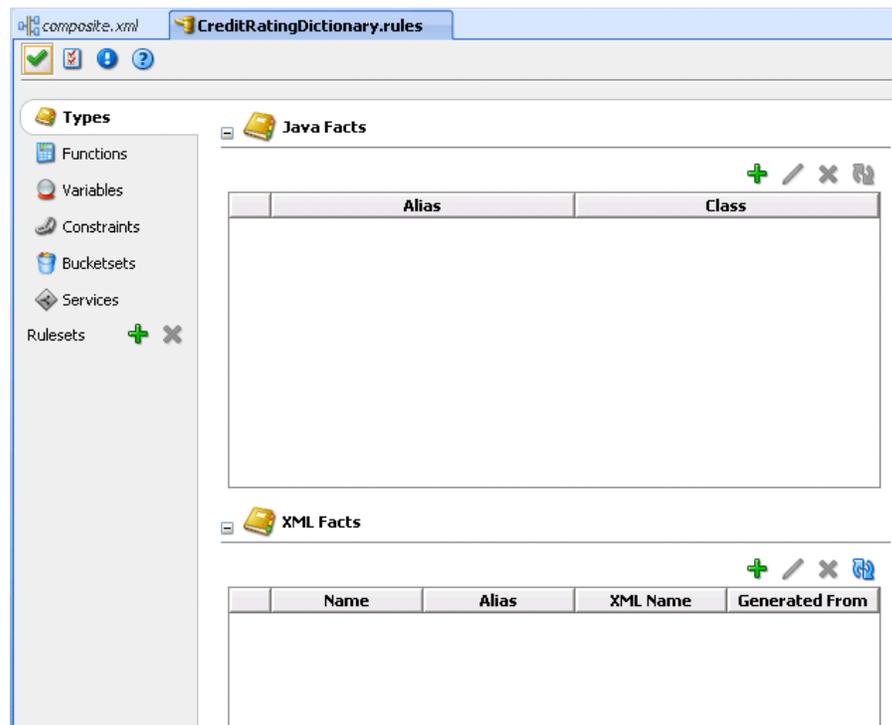
1. Go to the SOA Composite Editor.
2. Drag and drop a **Business Rule** from the **SOA** section of the **Component Palette** into the designer.
3. Enter a name in the **Dictionary Name** field or click the **Import** icon to import an existing dictionary into this component. The dictionary name becomes part of the `.decs` business rule metadata file name (for example, `CreditRatingDictionary.decs`).
4. Provide a package name for your business rule.
5. Accept the default path or specify a different path in which to create the business rule service component files.



6. Click **OK**.
This creates a business rule service component.



7. Double-click the business rule service component.
This displays the Rule Designer.



8. Create and edit business rules as necessary. The rules you create in this editor are added to the *business_rule_name.rules* file. Business rules modeling is a multistep process. For example, you can:
 - Specify the rules data model:
 - Import XML-schema definitions for XML-fact declarations
 - Create variables that are required for rule modeling
 - Create helper RL functions that can be used during rule modeling
 - Create Rulesets:
 - A ruleset can be considered as an execution unit
 - A ruleset comprises of one or more rules
 - Create Services:
 - Expose a ruleset as a service
 - Expose a RL function as a service

See Also: The following documentation for details about using the Rule Designer:

- *Oracle Fusion Middleware User's Guide for Oracle Business Rules*
- Tutorial included in the `soa-samples.zip` file

25.4.2 Business Rule Activity of a Business Process

If you want, you can associate a business rule service component created in the SOA Composite Editor with a BPEL process service component. You create this association with the business rule activity of the BPEL process. This activity creates a business rule partner link. This activity also enables you to create copy operation assignments between the fact data in your rule set or function and BPEL variables.

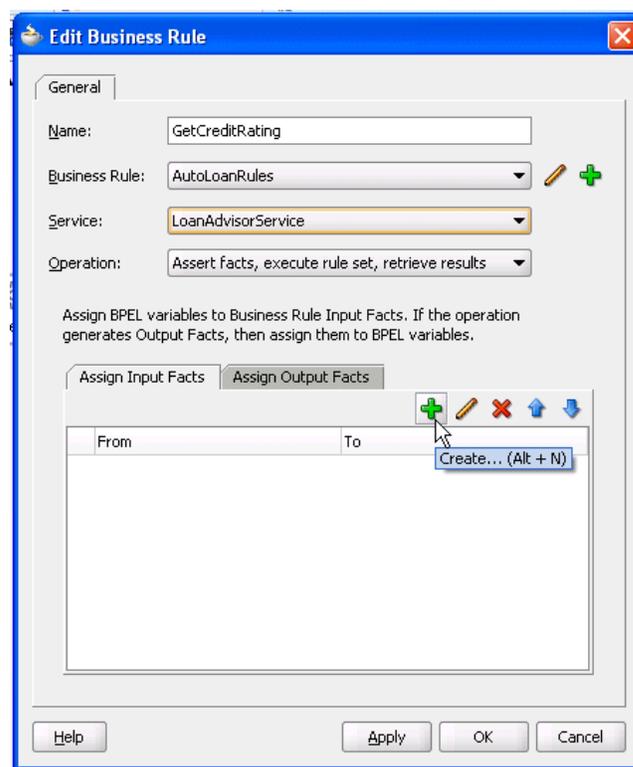
When complete, a business rule activity is created that consists of assign and invoke activities to the business rule partner link.

1. Go to the SOA Composite Editor.
2. Drag and drop a **BPEL Process** from the **SOA** section of the **Component Palette** into the designer.
3. Double-click the BPEL process in the SOA Composite Editor.
4. Select **BPEL** from the **Component Palette**.
5. Expand **BPEL Activities and Components**.
6. Drag and drop a **Business Rule** activity into your BPEL process.
The Edit Business Rule window appears.
7. Enter a name for the activity. When complete, this name becomes the partner link name.
8. Select the business rule service component you created in the Rule Designer in the SOA Composite Editor in [Section 25.4.1, "Business Rule Service Component"](#) on page 25-14. If you have not created a business rule, click the **Create** icon to the right of the **Business Rule** field to open the Create Business Rule window.
9. Select the service name that you specified in the Rule Designer.
10. Select the operation to perform that you specified in the Oracle Business Rules Designer.
 - If you created rule sets in the Rule Designer:
 - **Assert facts only** — Select the Rules Engine facts you want to assert (send factual data to the Rules Engine) in the future. You assign the required data for the facts with a BPEL assign activity. The underlying rule session must be stateful. Otherwise, the asserted facts are not visible to subsequent Rules Engine invocations.
 - **Assert facts and execute rule set** — The same as **Assert facts only**, except that the rule set is executed after the facts are asserted. A stateful rule session is created (or used). Otherwise, the result of executing this pattern is lost. No results are retrieved from the business Rules Engine.
 - **Retrieve results** — Retrieve a result from the business Rules Engine. The values of these results may have changed by past execution of a rule set acting on these facts. The activity assumes that it has a stateful rule session in its cache from which a result can be retrieved. This is the case if the

invocation pattern **Assert facts and execute rule set** operation was executed before in the BPEL process.

- **Assert facts, execute rule set, retrieve results, and reset the session** — The same as **Assert facts, execute rule set, and retrieve results**, except that the results are reset for the next time that you invoke the Web service. Resetting the session clears the previously asserted fact values.
- **Assert facts, execute rule set, and retrieve results** — The same as **Assert facts and execute rule set**, except that the results are retrieved from the business Rules Engine. You map the results of rule set execution to BPEL variables with an assign activity. The rules session remains active. This enables you to reuse previously asserted facts.
- If you created functions in the Rule Designer:
 - **Execute function** — Executes a function. Functions are also defined in dictionaries. For rule sets, you select input and output facts. For functions, there are a fixed set of input parameters and a single return value.
 - **Execute function and reset the session** — The same as **Execute function**, except that a stateful rule session is created for this pattern. All fact values are reset after retrieving the return value of the function.

11. Click **Assign Input Facts**, then click the **Create** icon to create mappings for the input facts.



This enables you to create assignments that map BPEL input variables to automatically created BPEL variables that correspond to the input fact type specified in the Rule Designer.

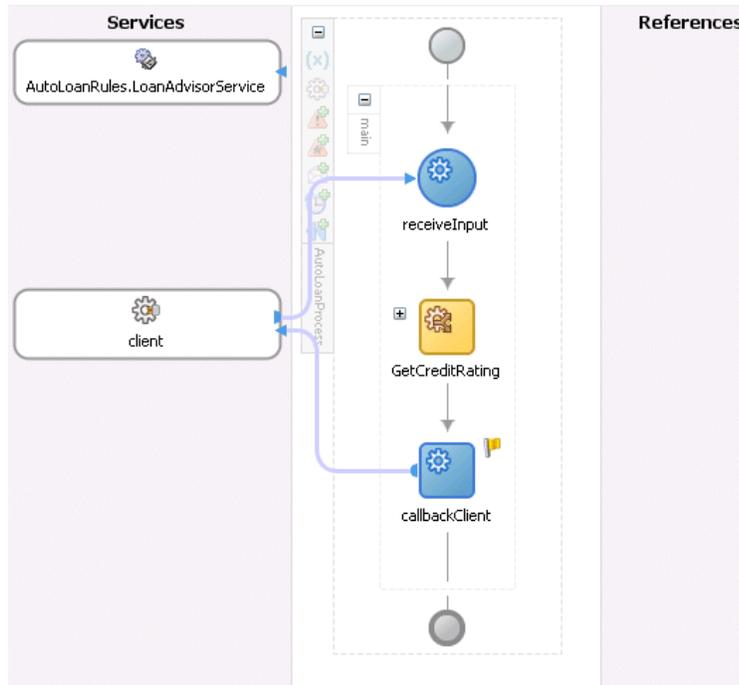
12. If you selected an invocation pattern that retrieves results, click **Assign Output Facts**, then click the **Create** icon to create mappings for the output facts.

This enables you to create assignments that map automatically created BPEL variables that correspond to the output fact type specified in the Rule Designer.

13. Click **OK** when complete.

This creates:

- A new business rule partner link for this Web service that interfaces with the Rules Engine.
- A business rule activity consisting of assign and invoke activities to the business rule partner link.
- A WSDL file based on the rule set is generated.



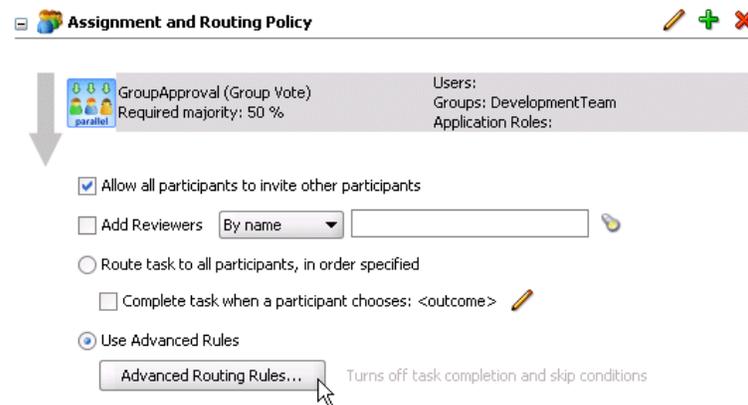
Business rule service component association with the BPEL process component is now complete.

25.4.3 Human Task Component

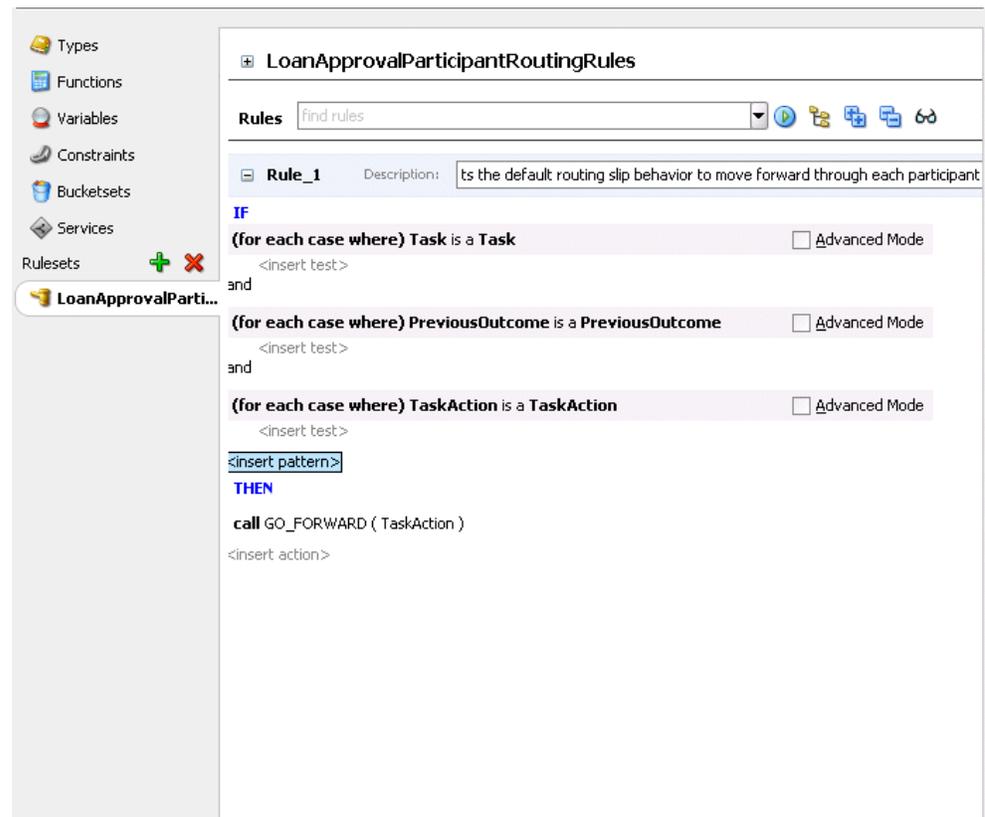
You can define state machine routing rules using Oracle Business Rules. This enables you to create business rules that are evaluated:

- After a task participant sets the outcome of a task
- Before the task is assigned to the next participant

This enables you to override the standard task routing method described in [Section 26.6.5.10, "Routing Tasks to All Participants in the Specified Order"](#) on page 26-40 and build complex routing behavior into tasks. You create business rules by clicking **Advanced Routing Rules** in the **Assignment and Routing Policy** section of the Human Task editor, as shown in [Figure 25-3](#).

Figure 25–3 Business Rule Integration with a Human Task Component

This starts the Rule Designer.

Figure 25–4 Rule Designer

See Also: [Section 26.6.5.12, "Advanced Task Routing Using Business Rules"](#) on page 26-42

25.5 Deploying a Business Rule

Business rules are deployed as part of the SOA composite application for which you create a deployment profile in Oracle JDeveloper.

See Also: [Section 3.4, "Deploying Applications"](#) on page 3-2 for instructions on creating and deploying a deployment profile in Oracle JDeveloper

25.6 Running a Business Rule in a SOA Composite Application

1. Log into Oracle Enterprise Manager Fusion Middleware Control.
`http://hostname:8888/SOAConsole`
2. Create and run a test instance of the SOA composite application that includes the business rule. See [Section 3.5, "Testing Applications"](#) on page 3-8 for instructions on creating and running a test instance.

This creates an instance of the SOA composite application.

3. Refresh the page after instance creation.
4. Click the instance ID of the SOA composite application in the **Last 5 Instances** section.

The screenshot displays the Oracle Enterprise Manager Fusion Middleware Control interface for the 'OrderProcessing (1.0)' composite application. The 'Instances and Faults' section shows 'Recent Faults and Rejected Messages' with 'No Faults found'. The 'Components' table lists various components with their types and instance counts. The 'Last 5 Instances' table shows two instances with Instance IDs 66 and 2, both with 0 faults. The 'Services and References' table lists Client, RapidService, FedExShipment, and CreditValidationService.

Name	Component Type	Total Instances	Active Instances	Completed Instances	Faults Count	Ave
ApproveOrder	Task	0	0	0	0	0
OrderFulfillment	Mediator	0	0	0	0	0
SelectManufacturer	BPEL	0	0	0	0	0
ApprovalRequired	Decision	0	0	0	0	0
Shipment	Mediator	0	0	0	0	0
OrderProcessor	BPEL	0	0	0	0	0

Instance ID	Faults	Start Time
66	0	7/19/07 9:32:02 PM
2	0	7/19/07 9:10:21 PM

Name	Faults Count	Average Processing Time
Client	0	0
RapidService	0	0
FedexShipment	0	0
CreditValidationService	0	0

The **Trace** section displays the flow of service components and service and reference binding components in the SOA composite application.

5. Click a business rule service component to view the flow trace of the component (for this example, **ApprovalRequired**).

Flow Trace Data Refreshed 23-Jul-07 06:58:02 PDT
 This page shows the flow of the message through various composite and component instances ECID 140.87.9.208:40026:1184905922554:0:461

Faults
 Click on a fault message to locate that fault in the trace view below

Message	Type	Time	At	Composite Instance	Application
No Faults Found					

Trace
 Click on a component instance to see its detailed audit trail
 Show Instance IDs

Instance	Type	Status	Time	Composite Instance	Application
Client	Service	Completed Successfully	19-Jul-07 21:32:02 PDT	OrderProcessing of 66	OrderBookingApp
OrderProcessor	BPEL Component	Completed Successfully	19-Jul-07 21:32:15 PDT	OrderProcessing of 66	OrderBookingApp
OrderSequence	Reference	Completed Successfully	19-Jul-07 21:32:07 PDT	OrderProcessing of 66	OrderBookingApp
Order	Reference	Completed Successfully	19-Jul-07 21:32:07 PDT	OrderProcessing of 66	OrderBookingApp
CustomerService	Reference	Completed Successfully	19-Jul-07 21:32:08 PDT	OrderProcessing of 66	OrderBookingApp
CreditValidatingService	Reference	Completed Successfully	19-Jul-07 21:32:08 PDT	OrderProcessing of 66	OrderBookingApp
ApprovalRequired	Decision Service Component	Completed Successfully	19-Jul-07 21:32:10 PDT	OrderProcessing of 66	OrderBookingApp
RapidService	Reference	Completed Successfully	19-Jul-07 21:32:11 PDT	OrderProcessing of 66	OrderBookingApp
SelectManufacturer	Reference	Completed Successfully	19-Jul-07 21:32:13 PDT	OrderProcessing of 66	OrderBookingApp
OrderFulfillment	Mediator Component	Completed Successfully	19-Jul-07 21:32:15 PDT	OrderProcessing of 66	OrderBookingApp
Shipment	Mediator Component	Completed Successfully	19-Jul-07 21:32:15 PDT	OrderProcessing of 66	OrderBookingApp
FedexShipment	Reference	Completed Successfully	19-Jul-07 21:32:15 PDT	OrderProcessing of 66	OrderBookingApp
USPSShipment	Reference	Completed Successfully	19-Jul-07 21:32:15 PDT	OrderProcessing of 66	OrderBookingApp
FulfillmentBatch	Reference	Completed Successfully	19-Jul-07 21:32:15 PDT	OrderProcessing of 66	OrderBookingApp
OrderStatus	Reference	Completed Successfully	19-Jul-07 21:32:15 PDT	OrderProcessing of 66	OrderBookingApp

This enables you to view flow trace and audit details for the business rule service component.

- Return to the page shown in Step 4 on page 25-20 and click a business rule service component shown in the **Components** table (for this example, **ApprovalRequired** is selected).

Applications
 OrderProcessing (1.0) Actions

Dashboard | Instances | **Faults and Rejected Messages** | Unit Test | Policies

Instances and Faults Instance Processing Time

Recent Faults and Rejected Messages Show more

Name	Instance ID	Type	Time	Source
No faults found				

Components

Name	Component Type	Total Instances	Active Instances	Completed Instances	Faults Count	Average
ApproveOrder	Task	0	0	0	0	0
OrderFulfillment	Mediator	0	0	0	0	0
SelectManufacturer	BPEL	0	0	0	0	0
ApprovalRequired	Decision	0	0	0	0	0
Shipment	Mediator	0	0	0	0	0
OrderProcessor	BPEL	0	0	0	0	0

Last 5 Instances Show more

Instance ID	Faults	Start Time
66	0	7/19/07 9:32:02 PM
21	0	7/19/07 9:10:21 PM

Services and References

Name	Faults Count	Average Processing Time
Client	0	
RapidService	0	
FedexShipment	0	
CreditValidatingService	0	

This page displays details about the completion success rate of business rule service component details.

The screenshot shows a web interface for the 'ApprovalRequired (Decision Service Component)'. It has a navigation bar with 'Dashboard', 'Instances', 'Faults', and 'Policies'. Below this are two expandable sections: 'Messages and Faults' and 'Average Processing Time'. The 'Messages and Faults' section is expanded to show 'Last 5 Faults', which contains a table with columns 'Message', 'Fault ID', 'Instance ID', and 'Type'. The message 'No Faults Found' is displayed. Below this is another expanded section 'Last 5 Instances', which contains a table with columns 'Instance ID', 'State', 'Faults', 'Start Time', and 'Last Modified'. Two instances are listed, both in a 'Completed Successfully' state with 0 faults.

Message	Fault ID	Instance ID	Type
No Faults Found			

Instance ID	State	Faults	Start Time	Last Modified
decision:95427a26-fe5	Completed Successfully	0	19-Jul-07 21:32:09 PDT	19-Jul-07 21:32:10 PDT
decision:318c2ac3-37c	Completed Successfully	0	19-Jul-07 21:10:30 PDT	19-Jul-07 21:10:34 PDT

- Click the instance IDs takes to view flow trace details.

Human Workflow Service Component

This part describes the human workflow service component.

This part contains the following chapters:

- [Chapter 26, "Designing Human Tasks"](#)
- [Chapter 31, "Designing Task Display Forms for Human Tasks"](#)
- [Chapter 28, "Human Task Services"](#)
- [Chapter 33, "Using Oracle BPM Worklist"](#)
- [Chapter 30, "Human Task and Approval Management Integration"](#)
- [Chapter 31, "Human Task and Microsoft Excel Integration"](#)

Designing Human Tasks

A company's business processes drive the integration of systems and people that participate in it. The business process and associated systems have a life cycle and certain behavior. The users who participate in the business process have roles and privileges to perform tasks in the business process. Using the workflow services of Oracle BPEL Process Manager, you can blend the integration of systems and services with human workflow into a single end-to-end process flow, while providing visibility and enabling exception handling and optimization at various levels.

This chapter contains the following topics:

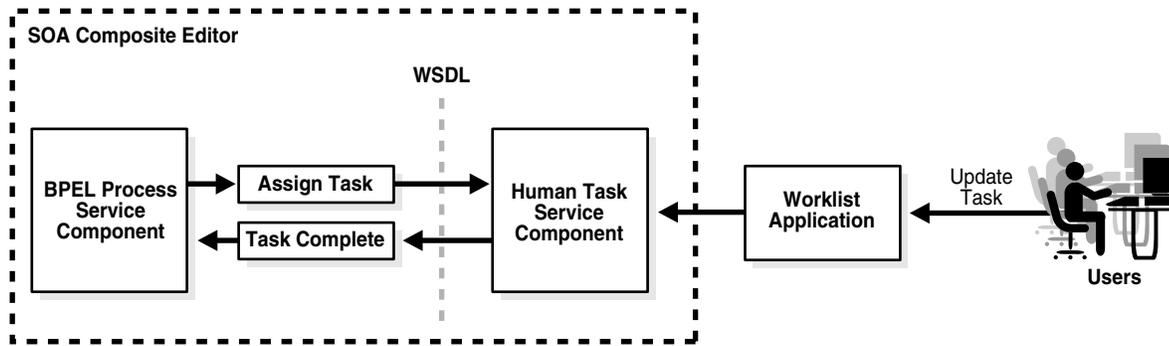
- [Section 26.1, "Introduction to Workflow Services"](#)
- [Section 26.2, "Use Cases for Workflow Services"](#)
- [Section 26.3, "Workflow Services Components"](#)
- [Section 26.4, "Participant Types in Workflow Services"](#)
- [Section 26.5, "Introduction to the Modeling Process"](#)
- [Section 26.6, "Creating the Human Task Definition with the Human Task Editor"](#)
- [Section 26.7, "Associating the Human Task Service Component with a BPEL Process"](#)
- [Section 26.8, "End-to-End Workflow Examples"](#)

See Also: [Chapter 2, "Introduction to the SOA Composite Editor"](#) for details about creating a human task service component in the SOA Composite Editor

26.1 Introduction to Workflow Services

Workflow services enable you to interleave human interactions with connectivity to systems and services within an end-to-end process flow. As shown in [Figure 26-1](#), workflow services are typically linked to a BPEL process service component through a WSDL contract, like any other Web service. The process assigns a task to a user or role and waits for a response. The users act on the task using Oracle BPM Worklist.

The human workflow service is responsible for handling all interactions with users or groups participating in the business process. It does this by creating and tracking tasks for the appropriate users in the organization. Users typically access tasks through a variety of clients, including the worklist application, e-mail, portals, or custom applications.

Figure 26–1 High-Level View of Workflow Services in Oracle BPEL Process Manager

Terms used in workflow services include:

- Task—work that needs to be done by a user, role, or group. In some cases, tasks are also referred to as work items.
- Notification—an e-mail, voice, instant message, or short message service (SMS) message that is sent when a user is assigned a task or informed that the status of the task has changed
- Worklist application—an enumeration of the tasks, or work items, assigned to or of interest to a user
- SOA Composite Editor — A tool that enables you to create SOA composite applications consisting of components such as BPEL processes, human tasks, business rules, and Oracle Mediator routing services.
- Human Task editor—A tool that enables you to specify task settings such as task outcome, payload structure, task participants, assignment and routing policy, expiration and escalation policy, notification settings, and so on
- .task file —The metadata task configuration file that stores the task settings specified with the Human Task editor
- routing slip—Contains information about the flow pattern for the workflow, assignees, escalation policy, expiration duration, signature policy, sequence in which the participants interact in the task, and so on.

For this release, significant enhancements have been made to the workflow services in Oracle BPEL Process Manager. Key workflow features include the following:

- Systems and services and human workflow integrations are blended into a single BPEL-based process flow.
- Multiple workflow patterns:
 - Many out-of-the-box patterns (participant types) including simple approval, sequential approval, parallel approval, adhoc workflow, FYI tasks, and so on
 - Support for variations such as automatic escalation, renewal, and reminders
 - Human Task editor includes wizards to create and configure patterns
 - Ability to mix and match built-in patterns to create complex patterns
 - Use business rules to define complex workflow patterns

See Also: [Section 26.6, "Creating the Human Task Definition with the Human Task Editor"](#) on page 26-12

- Task Assignment and routing:
 - Model human interactions as part of a business process and assign to users, roles, or groups
 - Support for task expiration, due dates, automatic renewal, and so on
 - Support for task delegation, escalation, and reapproval.
 - Storage of task history information for auditing purposes
 - Expression- or message-based task routing rules as well as adhoc routing
 - Pluggable assignment server
 - Ability to define task access control policies — what can be seen and edited by task creator, assignees, reviewer, and administrator

See Also: [Section 26.6, "Creating the Human Task Definition with the Human Task Editor"](#) on page 26-12
- Sophisticated task forms:
 - Automatically-generated task forms
 - Integration with Oracle application development framework (ADF) for custom forms using Faces technology
 - Integration with Microsoft Excel for initiating and acting on tasks

See Also: [Chapter 31, "Designing Task Display Forms for Human Tasks"](#)
- Worklist Application:
 - An out-of-the-box fully customizable worklist
 - Support for various user profiles — end user, supervisor, process owner, group owner, and administrator
 - Ability to perform authorized actions on tasks in the worklist, acquire and check out shared tasks, define personal to-do tasks, and define subtasks
 - Ability to filter tasks in a worklist view based on various criteria
 - Work queues — standard work queues such as high priority tasks, tasks due soon, and so on. Ability to define custom work queues.
 - Users can define custom vacation rules and delegation rules
 - Group owners can define task dispatching rules for shared tasks
 - Proxy access to selected subset of a user's worklist
 - Complete workflow history and audit trail
 - Digital signatures for tasks
 - Internationalization support — out-of-the-box support for eight languages

See Also: [Chapter 33, "Using Oracle BPM Worklist"](#)
- Workflow reports:
 - Out-of-the-box reports — productivity report, tasks priority report, cycle time report, and assignee time distribution report

- Workflow performance and error statistics in Oracle Enterprise Manager Fusion Middleware Control Console

See Also: [Chapter 33, "Using Oracle BPM Worklist"](#)

- Notification services:
 - Send notifications to specified users upon changes to specified tasks
 - Notifications through different delivery channels (e-mail, phone, instant message, and SMS)
 - Ability to customize content of the notifications for different types of tasks
 - Perform actions on tasks through e-mail
 - Guaranteed delivery of messages — support for bad e-mail format detection and blacklisted addresses (for incoming messages)

See Also:

- [Chapter 23, "Notifications and the Oracle User Messaging Service"](#)
- [Section 28.1.6, "Notification Service"](#) on page 28-11

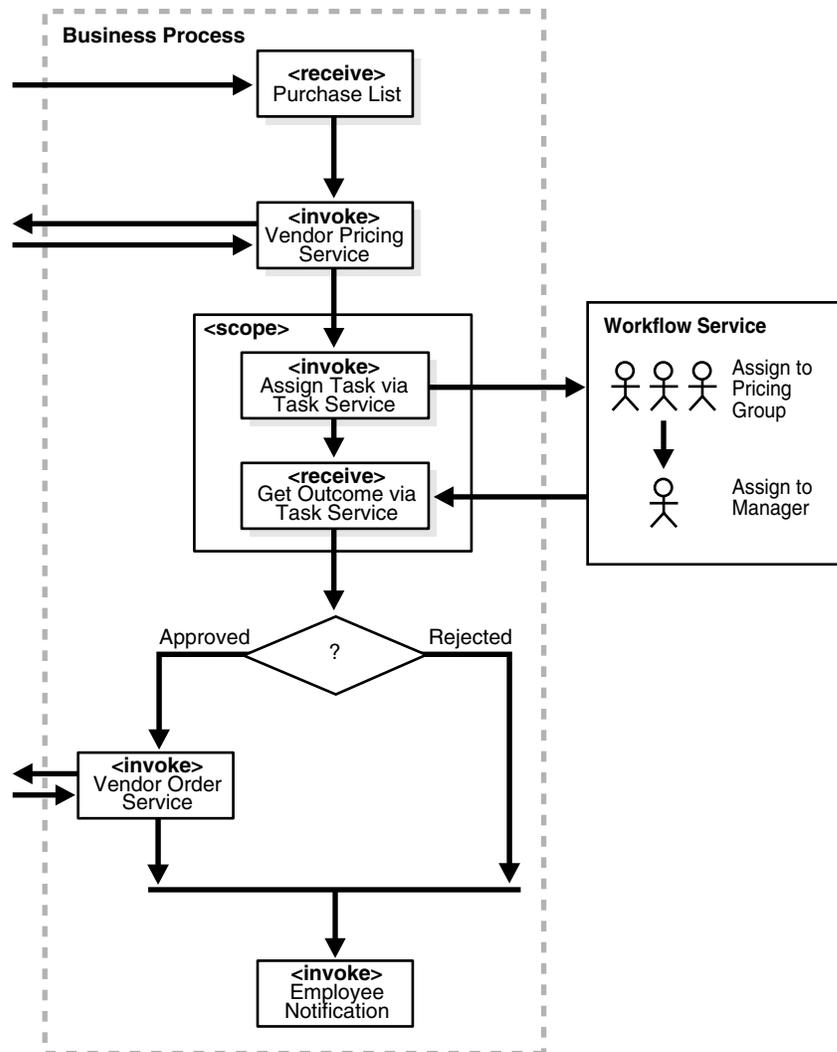
- Identity Services:
 - Role-based access control — assign permissions to roles and link an organization hierarchy to the role model for authorization
 - Assign worklist privileges to users, groups, and roles
 - Maintain user properties such as name, location, phone, and e-mail and capture organizational hierarchy (reporting structure) and group information
 - Integration with standard (for example, LDAP-based) directory services for user and role provisioning

See Also: [Section 28.1.5, "Identity Service"](#) on page 28-8

26.1.1 Workflow Functionality: A Procurement Process Example

The functionality of workflow services can be illustrated using a simple order approval business process to approve or reject an order, as shown in [Figure 26–2](#). requested items. Approval and rejection is a two-step process involving an initial approver and the manager of the initial approver. The order is first assigned to the Supervisor role. Once a user belonging to the Supervisor role approves the order, it is sent to this user's manager for final approval.

Figure 26–2 BPEL Workflow



26.2 Use Cases for Workflow Services

Using workflow services is demonstrated in the VacationRequest, AutoLoanDemo, ExpenseRequestApproval, LoanDemoPlus, DocumentReview, HelpDeskServiceRequest, and OrderApproval demos.

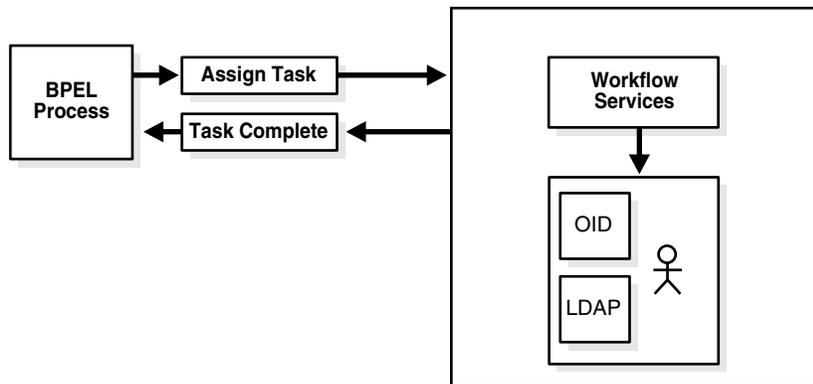
See Also:

- [Section 26.8, "End-to-End Workflow Examples"](#) on page 26-81
- `SOA_Oracle_Home\bpel\samples\demos`

The following sections describe multiple use cases for workflow services.

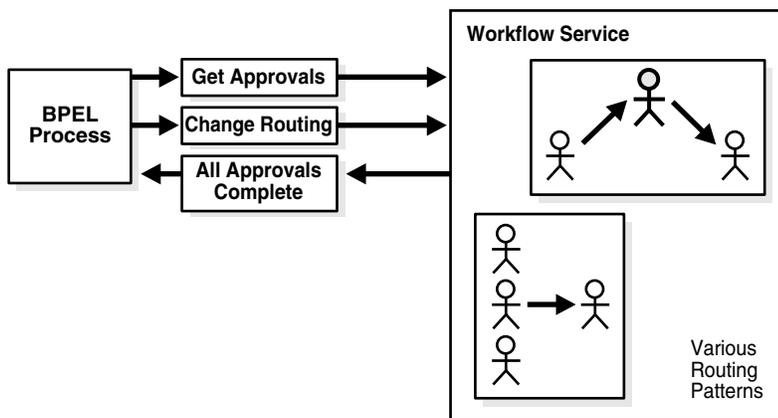
26.2.1 Assigning a Task to a User or Role

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 store. This scenario, shown in [Figure 26–3](#), is described in the OrderApproval sample.

Figure 26–3 Assigning Tasks to a User or Role from a Directory

26.2.2 Using 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, shown in [Figure 26–4](#), is described in the LoanDemoPlus sample.

Figure 26–4 Flow Patterns and Routing Policies

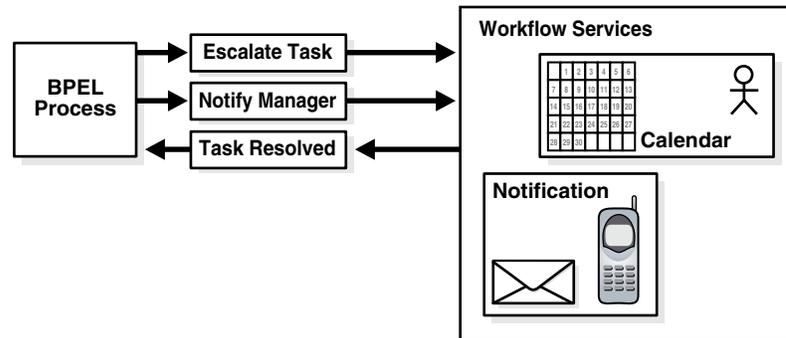
See [Section 26.4, "Participant Types in Workflow Services"](#) on page 26-10 for the various flow types supported by workflow services. You can use these types as building blocks to create complex workflows.

26.2.3 Escalation, Expiration, and Delegation

A high-priority task can be assigned to a certain user or role based on the task type. 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 e-mail, 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 26–5](#). For example, consider a support call, which is part of the HelpDeskServiceRequest

process. A high-priority task may be assigned to a certain user and if the user does not respond in two days, then the task is routed to the manager for further action.

Figure 26–5 Escalation and Notification



26.2.4 Automatic Assignment and Delegation

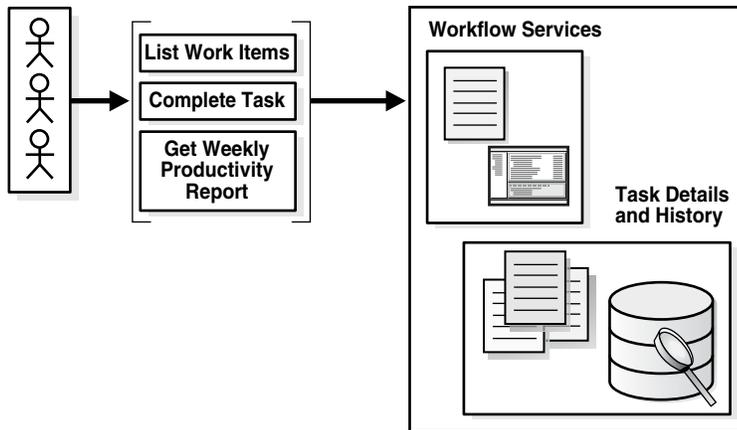
A user may decide to have another user perform tasks on their behalf. Tasks can be explicitly delegated from the Oracle BPM Worklist or can be automatically delegated. For example, a manager sets up a vacation rule saying that all their high priority tasks are automatically routed to one of their reports 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).

26.2.5 Work Queues and Proxy Support

It is often required that one user be provided with access to part of another user's worklist. For example, an executive decides to provide access to expense approvals within a certain limit to their secretary. Work queues allow you to create a custom view to group a subset of tasks in the worklist (say high priority tasks, tasks due in 24 hours, expense approval tasks, and so on). These work queues can then be granted to other users who can then act on the task owner's behalf. For example, in the scenario described above, the executive can create a delegated expense approvals work queue for expenses below \$5000.

26.2.6 The Oracle BPEL Worklist Application

Users typically access tasks assigned to them by using the Oracle BPM Worklist, as shown in [Figure 26–6](#). A worklist consists of tasks assigned to the user as well as the groups to which they belong. A task may also include forms and attachments in addition to other task details such as history, comments, and approval sequence. The worklist may also be accessed from Oracle Portal or other clients to act on tasks as well as get productivity reports. The Oracle BPM Worklist can be customized and extended based on the specific needs of an application. See [Chapter 33, "Using Oracle BPM Worklist"](#) for details about worklist functionality and the sample Oracle BPM Worklist.

Figure 26–6 Oracle BPM Worklist—Access Tasks, Forms, Attachments, and Reports

26.3 Workflow Services Components

Starting with release 11g, all human task metadata is stored and managed in the Metadata Service (MDS) repository. The workflow service consists of a number of services that handle various aspects of human interaction with a business process.

Figure 26–7 shows the following workflow services 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: e-mail, telephone voice message,

instant message, and SMS. See [Section 28.2, "Notifications from Human Workflow"](#) on page 28-20 for more information.

- User Metadata Service

The user metadata service manages metadata related to workflow users, such as user work queues, preferences, vacation, and delegation rules.

- Runtime config service

The runtime config service provides methods for managing metadata used in the task service run time 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 26–7 Workflow Services Components

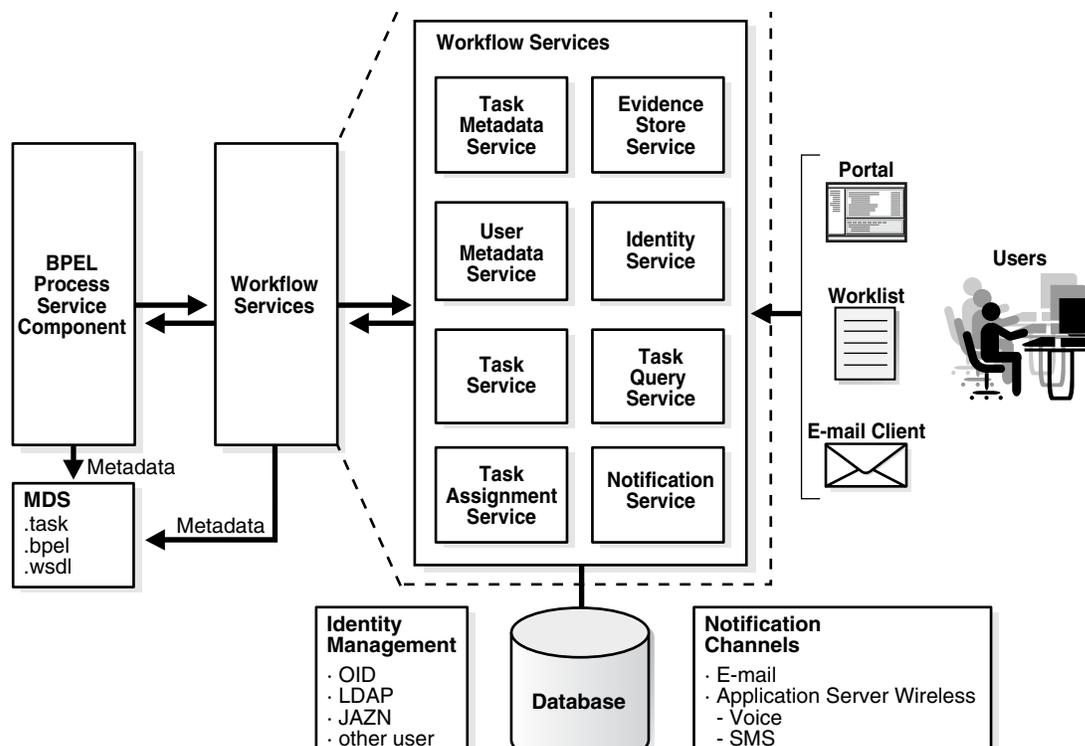
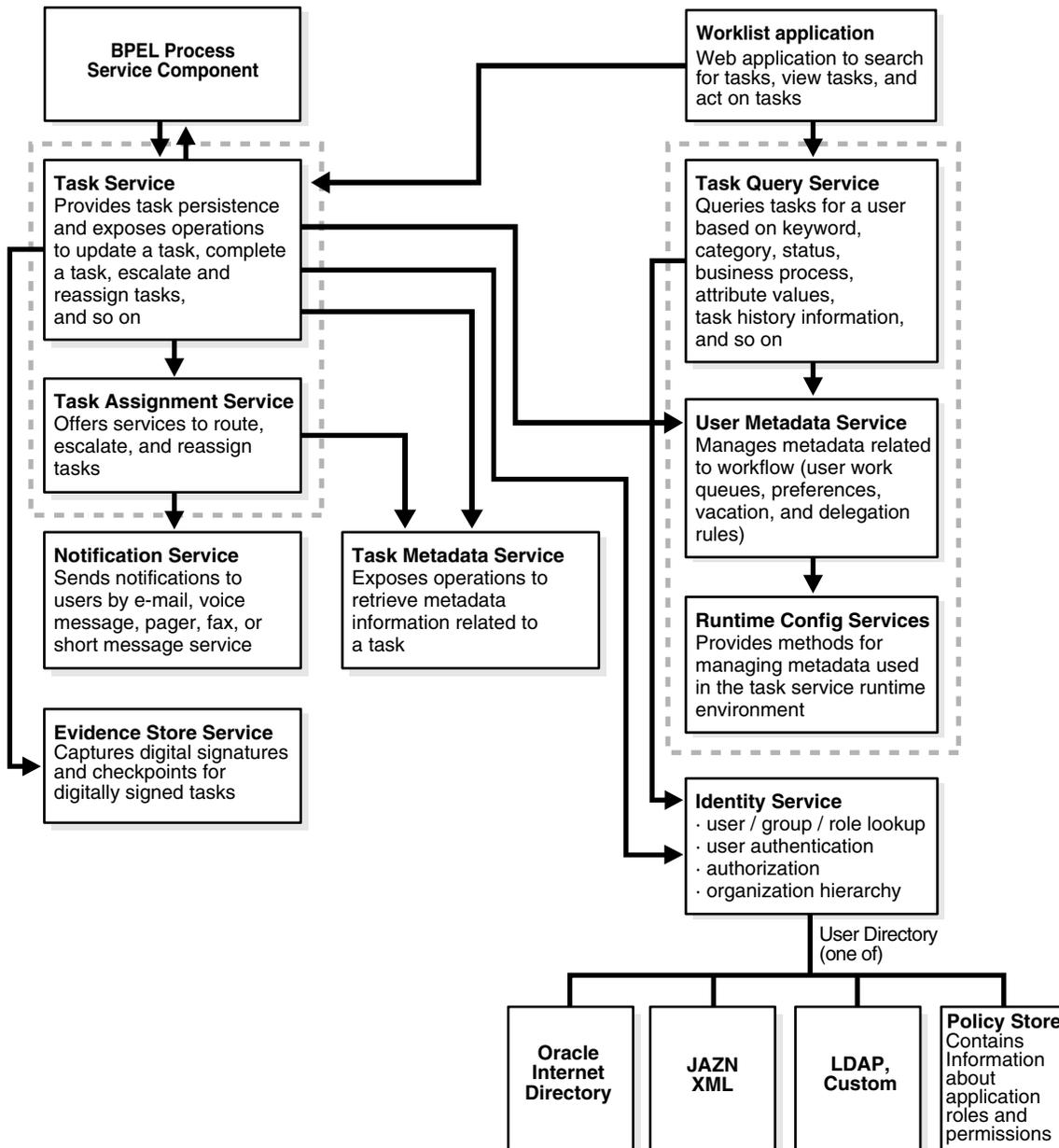


Figure 26–8 shows the interactions between the services and the business process.

Figure 26–8 Workflow Services and Business Process Interactions



26.4 Participant Types in Workflow Services

Oracle BPEL Process Manager provides a library of participant types (known in pre-10.1.3 releases as workflow patterns). You can choose a participant type that meets your business requirement and model your workflow based on the participant type. Oracle BPEL Process Manager supports the following participant types:

- Single Approver — used 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. Based on the user's action, you define what the business process does.
- Group Vote — used when multiple users, working in parallel, must take action 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.

- **Management Chain**—used to route tasks for approval to multiple users in a management chain hierarchy. You specify the task participants as a management chain list or a list of users.
- **Sequential list of approvers (extension of a sequential workflow)**—used to create a list of sequential participants for a workflow. This type is similar to the management chain participant type, except that with that type, the users are part of an organization hierarchy. For the sequential list of approvers participant type, they can be any list of users or groups.
- **FYI assignee** — used when a task is sent to a user, but the business process does not wait for a user response; it just continues. FYI assignees cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.
- **External Routing Service** —used to configure an external routing service that dynamically determines the participants in the workflow. If this participant type is specified, all other participant types are ignored. It is assumed that the external routing service provides a list of participant types (single approver, list of approvers, group vote, and so on) at run time to determine the routing of the task.
- **Rule-based assignments** — in addition to the built-in patterns, you can use the state-based routing rules to create complex, dynamic workflows
- **Mixing and matching patterns** — mix and match patterns to create complex workflows

26.4.1 Chaining Multiple Tasks

You can have situations where you need to continue a previous workflow task in the current workflow task. Oracle BPEL Process Manager enables you to include the task history, comments, and attachments from the previous task. This provides you with a complete end-to-end audit trail.

See Also: [Section 26.7.4.4, "Including the Task History of Other Human Tasks"](#) on page 26-78

26.5 Introduction to the Modeling Process

The modeling process consists of creating and modeling a human task service component in the SOA Composite Editor, optionally associating it with a BPEL process, and generating the task form for displaying the human task during run time in the Oracle BPM Worklist.

This section provides a brief overview of these modeling tasks and provides references to specific modeling instructions.

- [Section 26.5.1, "Create a Human Task Definition with the Human Task Editor"](#)
- [Section 26.5.2, "Optionally Associate the Human Task Definition with a BPEL Process"](#)
- [Section 26.5.3, "Generate the Task Display Form"](#)

See Also: [Chapter 2, "Introduction to the SOA Composite Editor"](#) for details about using the SOA Composite Editor

26.5.1 Create a Human Task Definition with the Human Task Editor

The Human Task editor enables you to define the metadata for the task. This editor enables you to specify human task settings, such as task outcome, payload structure, task participants, 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.

See Also: [Section 26.6, "Creating the Human Task Definition with the Human Task Editor"](#) on page 26-12 for specific instructions

26.5.2 Optionally Associate the Human Task Definition with a BPEL Process

You can optionally associate the `.task` file that consists of the human task settings with a BPEL process. Association is made with a human task activity that you drag and drop into your BPEL process for configuring. You also define the task definition, task initiator, task priority, and map the task parameter that carries 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.

See Also: [Section 26.7, "Associating the Human Task Service Component with a BPEL Process"](#) on page 26-69 for specific instructions

26.5.3 Generate the Task Display Form

You generate the Oracle ADF layout of the task display form used for displaying the task details at run time in Oracle BPM Worklist.

See Also: [Chapter 31, "Designing Task Display Forms for Human Tasks"](#)

26.6 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, task participants, assignment and routing policy, expiration and escalation policy, notification settings, and so on.

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 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 JDeveloper

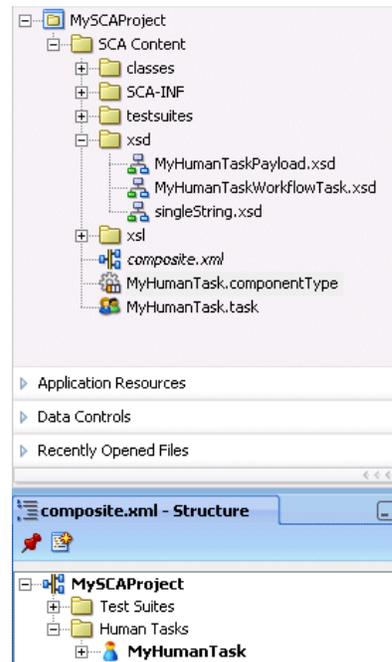
- The human task icon in the SOA Composite Editor or in your BPEL process in Oracle JDeveloper

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 26–9 shows these folders and files.

Figure 26–9 Human Task Folders and Files



This section contains the following topics:

- Section 26.6.1, "Creating a Human Task Service Component and Accessing the Human Task Editor"
- Section 26.6.2, "Reviewing the Sections of the Human Task Editor"
- Section 26.6.3, "Specifying the Task Title, Priority, Outcome, and Owner"
- Section 26.6.4, "Specifying the Task Payload Data Structure"
- Section 26.6.5, "Assigning Task Participants"
- Section 26.6.6, "Escalating, Renewing, or Ending the Task"
- Section 26.6.7, "Specifying Participant Notification Preferences"
- Section 26.6.8, "Specifying Advanced Settings"
- Section 26.6.10, "Exiting the Human Task Editor and Saving Your Changes"

26.6.1 Creating a Human Task Service Component and Accessing the Human Task Editor

You create a human task service component in the SOA Composite Editor. 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.

1. Go to the SOA project in which to create a human task service component in the SOA Composite Editor.
2. Select **SOA** from the **Component Palette**.
3. Drag and drop a **Human Task** from the list.

The Human Task Wizard - Project Settings window appears.

4. Enter a name in the **Human Task Definition Name** field.

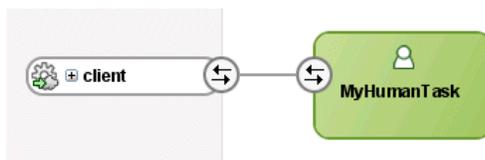
The name you enter is added as the `.task` file name in the directory path of the **Directory** field.

`C:\JDeveloper\jdev\mywork\my_application_name\sca_project_name\`

5. Note the **Expose as Composite Service** check box. You selection of this check box 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 **Expose as Composite Service** check box. The human task service component is created as a component that you explicitly associate with a BPEL process service component.



- b. If you want to create the human task service component as a standalone component in the SOA Composite Editor, select the **Expose as Composite Service** check box. This creates a human task service component that is automatically wired to a SOAP adapter service:



This service provides external customers with an entry point into the human task service component of the SOA composite application.

6. Click **Finish**.

The **Human Task** icon appears in the SOA Composite Editor canvas workspace and a **Human Task** folder appears in the **Structure** window in the lower left section of the SOA Composite Editor.

7. Double-click the **Human Task** icon.

The Human Task editor appears.

8. Go to section [Section 26.6.2, "Reviewing the Sections of the Human Task Editor"](#) on page 26-15.

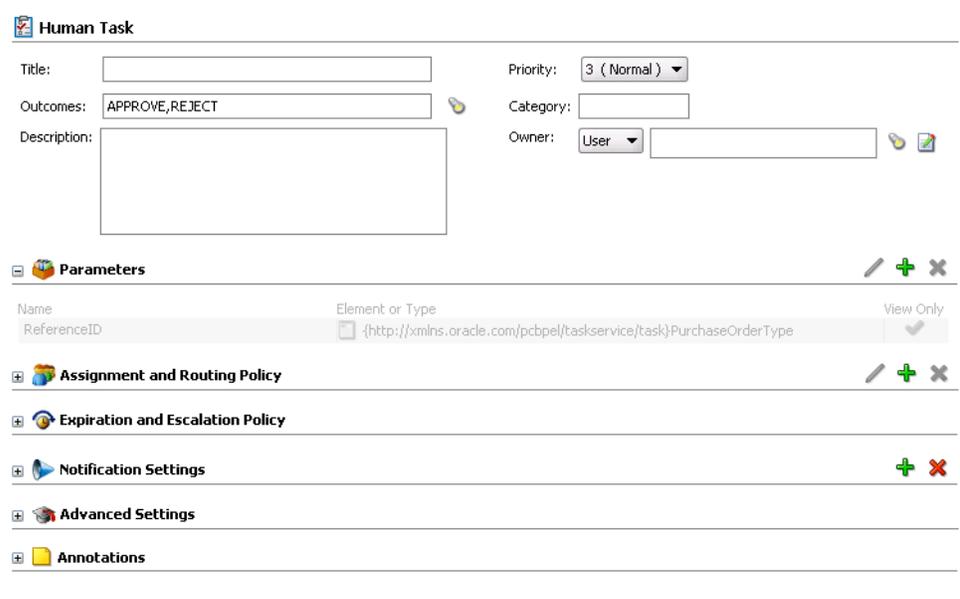
Note: You can also create a human task that you *later* associate with a BPEL process by selecting **New** from the **File** main menu, then selecting **SOA Tier > SOA Components > Human Task**.

See Also: [Chapter 2, "Introduction to the SOA Composite Editor"](#) for details about creating a human task service component in the SOA Composite Editor

26.6.2 Reviewing the Sections of the Human Task Editor

The Human Task editor consists of the following main sections shown in [Figure 26–10](#). These sections enable you to create a human task.

Figure 26–10 Human Task Editor



Instructions for using these main sections of the Human Task editor to create a workflow task are listed in [Table 26–1](#).

Table 26–1 Human Task Editor

For This Main Section...	See...
Task Configuration (title, outcomes, priority, and owner)	Section 26.6.3, "Specifying the Task Title, Priority, Outcome, and Owner" on page 26-16
Parameters	Section 26.6.4, "Specifying the Task Payload Data Structure" on page 26-21
Assignment and Routing Policy	Section 26.6.5, "Assigning Task Participants" on page 26-22
Expiration and Escalation Policy	Section 26.6.6, "Escalating, Renewing, or Ending the Task" on page 26-46
Notification Settings	Section 26.6.7, "Specifying Participant Notification Preferences" on page 26-53

Table 26–1 (Cont.) Human Task Editor

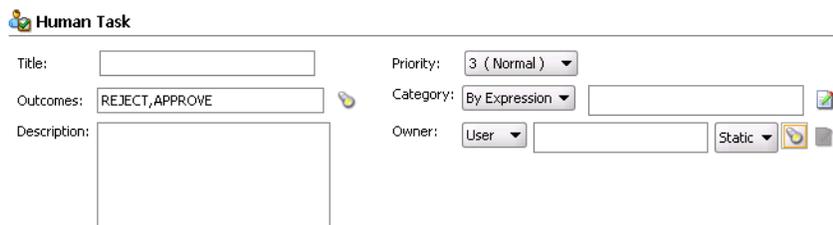
For This Main Section...	See...
Advanced Settings For specifying: <ul style="list-style-type: none"> ■ Custom escalation rules ■ Custom style sheets for attachments ■ Multilingual settings ■ Error messages ■ Callback classes ■ Workflow signature policies ■ Access rules to task content 	Section 26.6.8, "Specifying Advanced Settings" on page 26-57
Annotations	Section 26.6.9, "Specifying Annotations" on page 26-68

26.6.3 Specifying the Task Title, Priority, Outcome, and Owner

Figure 26–11 shows the **Task Configuration** section of the Human Task editor.

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

Figure 26–11 Human Task Editor — Task Configuration Section



Instructions for configuring the following subsections of the **Task Configuration** section are listed in Table 26–2:

Table 26–2 Human Task Editor — Task Configuration Section

For This Subsection...	See...
Title Priority	Section 26.6.3.1, "Specifying a Task Title and Priority" on page 26-16
Outcomes	Section 26.6.3.2, "Specifying a Task Outcome" on page 26-17
Description	Section 26.6.3.3, "Specifying a Task Description" on page 26-18
Category	Section 26.6.3.4, "Specifying a Task Category" on page 26-18
Owner	Section 26.6.3.5, "Specifying a Task Owner" on page 26-18

26.6.3.1 Specifying a Task Title and Priority

1. Enter the following details.

Field	Description
Title	Enter an optional task title. The task title displays in the Oracle BPM Worklist. If you enter a title in the Task Title field of the General tab of the Human Task window described in Section 26.7.2.1, "Specifying the Task Title" on page 26-71, the title you enter here is overridden.
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 . The priority can be used to sort tasks in the Oracle BPM Worklist. This priority value is overridden by any priority value you select in the General tab of the Add a Human Task window. See Also: Section 26.7.2.2, "Specifying the Task Initiator and Task Priority" on page 26-72 for instructions on specifying a priority value in the Add a Human Task window

26.6.3.2 Specifying a Task Outcome

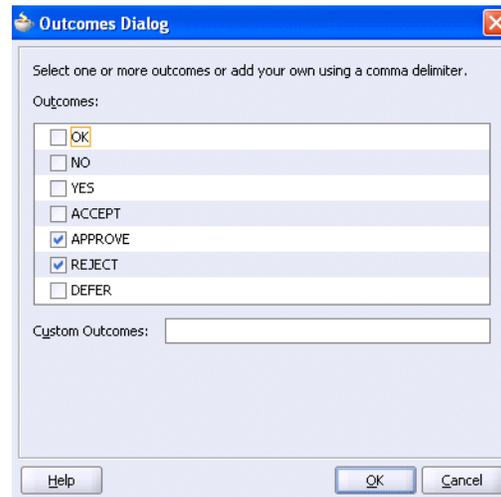
Task outcomes capture the possible outcomes of a task. The Oracle BPM Worklist displays the outcomes you specify here as the possible actions to perform during run time. You can specify the following types of task outcomes:

- Select a seeded outcome
- Enter a custom outcome

The task outcomes can also have run time display values that are different from the actual outcome value specified here. This permits outcomes to be displayed in a different language in the Oracle BPM Worklist. See [Section 26.6.8.4, "Specifying Multilingual Settings"](#) on page 26-59 for more information about internationalization.

1. Click the **Browse** icon to the right of the **Outcomes** field.

The Outcomes window displays the possible outcomes for tasks. **APPROVE** and **REJECT** are selected by default.



2. Select additional task outcomes or deselect the default outcomes.
3. Enter any custom outcomes separated by commas in the **Custom Outcomes** field.
4. 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 group vote participant type.

See Also: [Section 26.6.5.3.4, "Specifying Group Voting Details"](#) on page 26-30

26.6.3.3 Specifying a Task Description

You can optionally specify a description of the task in the **Description** field. The description does not display in the Oracle BPM Worklist.

26.6.3.4 Specifying a Task Category

You can optionally specify a task category in the **Category** field. This categorizes tasks created in a system. This displays in Oracle BPM Worklist.

26.6.3.5 Specifying a Task Owner

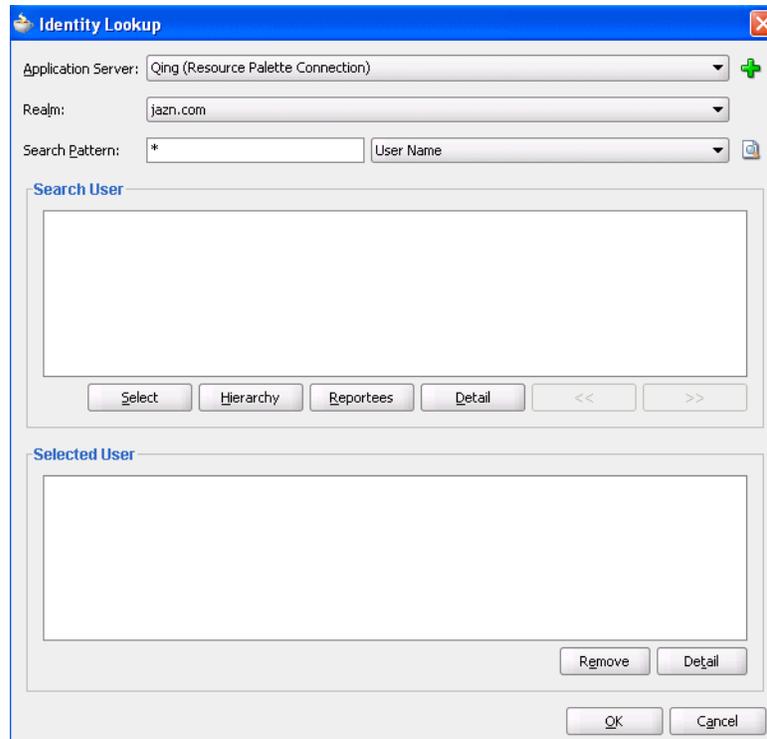
The task owner can view the tasks belonging to business processes they own and perform operations on behalf of any of the assigned task participant types. Additionally, the owner can also reassign, withdraw, or escalate tasks. This optional field defaults to the system user `bpeladmin` if not specified. The task owner can also be specified in the **Advanced** tab of the Human Task window described in [Section 26.7.4.2, "Specifying a Task Owner"](#) on page 26-78. The task owner specified in the **Advanced** tab overrides any task owner you enter here.

1. Select a method for specifying the task owner:
 - [Section 26.6.3.5.1, "Specifying a Task Owner By Browsing the User Directory"](#)
 - [Section 26.6.3.5.2, "Specifying a Task Owner Dynamically"](#)

26.6.3.5.1 Specifying a Task Owner By Browsing the User Directory

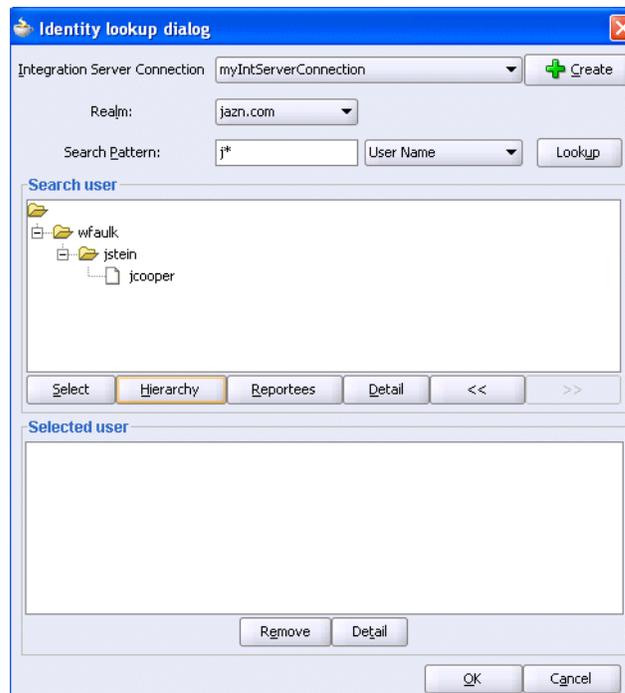
Task owners can be selected by browsing the user directory (Oracle Internet Directory (OID), Java AuthorizatioN (JAZN)/XML, LDAP, and so on) that is configured for use with Oracle BPEL Process Manager.

1. Click the first icon to the right of the **Owner** field to display the Identity Lookup dialog.
2. Search for the owner by entering a search string such as `jcooper`, `j*`, `*`, and so on. Clicking **Lookup** fetches all the users that match the search criteria.



One or more users or groups can be highlighted and selected by clicking **Select**.

3. 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.



4. View the details of a user or group by highlighting the user or group and clicking **Detail**.



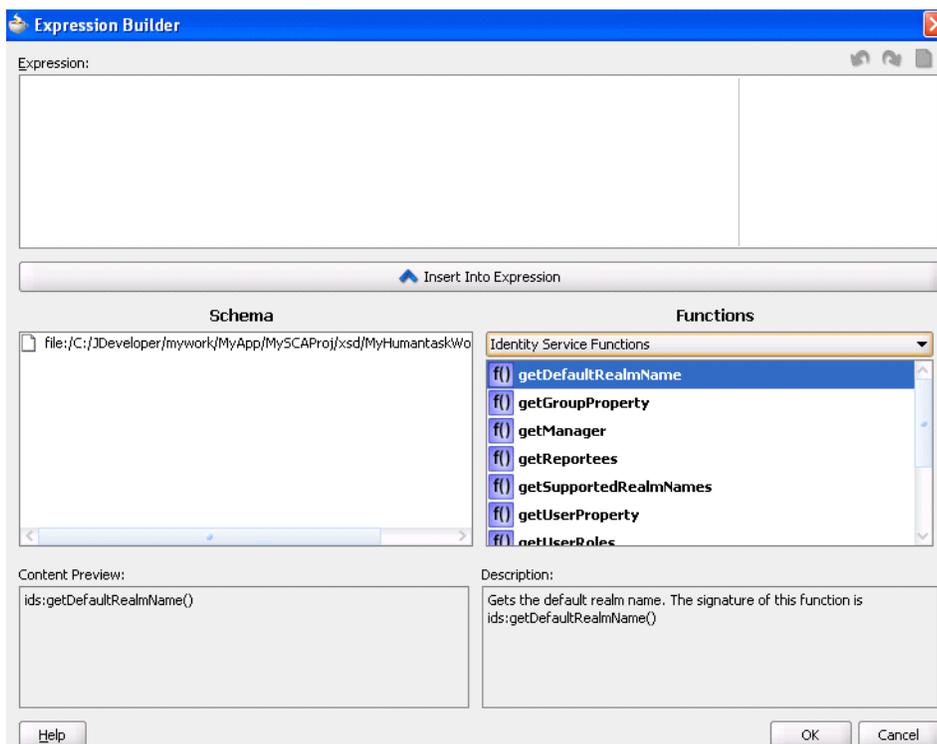
5. Click **OK** to return to the Identity Lookup dialog.
6. Click **Select** to add the user to the **Selected user** section.
7. Click **OK** to return to the Human Task editor.

Your selection displays in the **Owner** field.

26.6.3.5.2 Specifying a Task Owner Dynamically

Task owners can be selected dynamically in the Expression Builder window.

1. Click the second icon to the right of the **Owner** field to display the Expression Builder window:



2. Browse the available variable schemas and functions to create a task owner.
3. Click **OK** to return to the Human Task editor.

Your selection displays in the **Owner** field.

See Also:

- Click **Help** for instructions on using the Expression Builder window and XPath Building Assistant
- [Section 28.4, "Human Task Service and Identity Service Related XPath Extension Functions"](#) on page 28-36 for information about workflow service dynamic assignment functions and identity service functions

26.6.4 Specifying the Task Payload Data Structure

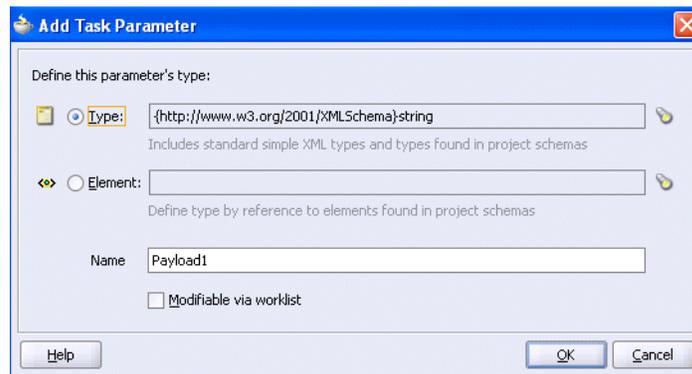
Figure 26–12 shows the **Parameters** section of the Human Task editor.

This section enables you to define the structure (message attributes) of the task payload (the data in the task). 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 26–12 Human Task Editor — Parameters Section



1. Click the + sign to display the Add Task Parameter window.



2. Enter the following details:

Field	Description
Parameter Type	Select Type or Element and click the Browse icon to display the Type Chooser window for selecting the task parameter.
Name	Accept the default name or enter a custom name. This field only displays if Type is the selected parameter type.
Modifiable via worklist	Select this check box to enable users to edit task payload data in the footer of the Oracle BPM Worklist. For example, the approver in the application may need to add approver comments.

Note: You can only define payload flex field mappings in the Oracle BPM Worklist for payload parameters that are simple XML types.

3. Click **OK** to return to the Human Task editor.
Your selection displays in the **Parameters** section.
4. If you want to edit your selection, highlight it and click the first icon in the upper right part of the **Parameters** section.

26.6.5 Assigning Task Participants

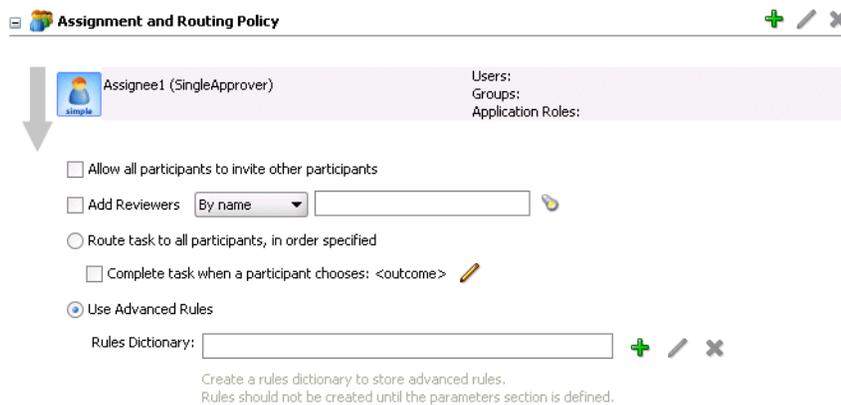
Figure 26–13 shows the **Assignment and Routing Policy** section of the Human Task editor.

This section enables you to select a participant type that meets your business requirement. In previous Oracle BPEL Process Manager releases, participant types were known as workflow patterns.

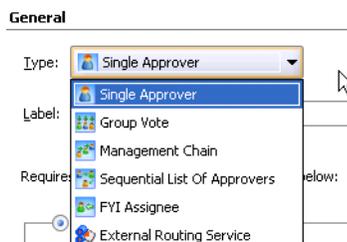
You can easily mix and match multiple participant types to model the human task to create a complex workflow routing policy. This enables you to extend the functionality of a previously configured human task to model more complex workflows.

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.

Figure 26–13 Human Task Editor — Assignment and Routing Policy Section



1. Click the + sign to display the Add Participant Type window.
This enables you to select a specific participant type.
2. Select a participant type from the **Type** list.



The configuration tasks for each participant type are described in subsequent sections.

3. See the following section based on your selection:

For This Subsection...	See...
Add Participant Type	
<ul style="list-style-type: none"> ■ Single approver ■ Group vote ■ Management chain ■ Sequential list of approvers ■ FYI assignee ■ External routing service 	<p>Section 26.6.5.2, "Configuring the Single Approver Participant Type" on page 26-24</p> <p>Section 26.6.5.3, "Configuring the Group Vote Participant Type" on page 26-27</p> <p>Section 26.6.5.4, "Configuring the Management Chain Participant Type" on page 26-31</p> <p>Section 26.6.5.5, "Configuring the Sequential List of Approvers Participant Type" on page 26-34</p> <p>Section 26.6.5.6, "Configuring the FYI Assignee Participant Type" on page 26-37</p> <p>Section 26.6.5.7, "Configuring the External Routing Service Participant Type" on page 26-38</p>

4. See the following task assignment and routing policy sections shown in [Figure 26–13](#) *after* you have configured a participant type. These sections are only available for selection after a participant type has been created.

For This Subsection...	See...
Allow all participants to invite other participants	Section 26.6.5.8, "Allowing All Participants to Invite Other Participants" on page 26-40
Add Reviewers	Section 26.6.5.9, "Adding Reviewers" on page 26-40
Route task to all participants, in order specified	Section 26.6.5.10, "Routing Tasks to All Participants in the Specified Order" on page 26-40
Complete task when a participant chooses <outcome>	Section 26.6.5.11, "Abruptly Completing a Condition" on page 26-41
Using Advanced Rules	Section 26.6.5.12, "Advanced Task Routing Using Business Rules" on page 26-42

26.6.5.1 Specifying Task Approvers

There are three types of task participants:

- Users
- Groups — These are sets of users or other groups or roles in the enterprise identity store (OID, LDAP, and so on)
- Application roles — These are application-specific roles. These define permissions that a certain set of users have with respect to a specific application. These are generally stored locally in the application server policy store and not in the enterprise identity store

Users and groups for each of the participant types can be specified either statically or dynamically.

When the users, groups, and application roles are specified 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`), which in the case of a single approver, is captured as follows:

```
<participant name="Assignee1">
  <resource isGroup="false" type="STATIC">jstein</resource>
</participant>
```

- A delimited string of users, groups, or application roles (for example, `jstein, wfaulk, cdickens`), which in the case of a single approver, is captured as follows:

```
<participant name="Assignee1">
  <resource isGroup="false" type="STATIC">jstein, wfaulk, cdickens</resource>
</participant>
```

You may have a business requirement to create a dynamic list of task approvers specified in a payload variable. This 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. For example, the following task shows that the payload message attribute is of type `xsd:String` and its value is a comma-delimited string of approvers. This node can be used to specify the participants.

```
<task>
  . . .
  <payload>
    <approvers>jstein,wfaulk,cdickens</approvers>
  </payload>
</task>
```

The default delimiter for the assignee delimited string is a comma (,). This delimiter can be changed using the `assigneeDelimiter` XML element in the `wf-config.xml` file. This delimiter applies to all workflows in the system.

Specifying participants in this manner is applicable to all participant types, although they are interpreted differently for each type. For example:

- In a single user participant type, the task is assigned to everyone evaluated.
- In a sequential list of approvers participant type, the task is sequentially assigned to users and groups evaluated in the list.
- In a group vote participant type, a task is created for each user and group evaluated in the list.

This interpretation of resource XPath expressions provides `orcl:create-nodeset-from-delimited-string-equivalent` functionality to enable you to specify a dynamic list of one or more task approvers (resource element members) from the payload variable.

26.6.5.2 Configuring the Single Approver Participant Type

Figure 26–14 displays the Single Approver window.

This participant type requires 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. Based on the user's action, you define what the business process does.

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.

Figure 26–14 Add Participant Type — Single Approver

Type:

Label:
e.g., Approval Manager

Requires action from **one** of the participants below:

By name By expression

User Id(s)

Enter comma separated names or browse for values.

Group Id(s)

Enter comma separated names or browse for values.

Application Role(s)

Enter comma separated names or browse for values.

Specify skip rule

Advanced

Limit allocated duration to:

Day Hour Minutes

Over all task duration is currently set to 0 days, 0 hours, and 0 minutes.

Allow this participant to invite other participants

1. Enter a recognizable label for this participant in the **Label** field. 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 Add Participant Type - Single Approver window are listed in [Table 26–3](#):

Table 26–3 Add Participant Type — Single Approver

For This Subsection...	See...
Requires action from one of the participants below	Section 26.6.5.2.1, "Assigning Participants to the Single Approver Task" on page 26-25
Specify skip rule	Section 26.6.5.2.2, "Bypassing a Task Participant" on page 26-26
Limit allocated duration to (under the Advanced section)	Section 26.6.5.2.3, "Specifying a Time Limit for Acting on a Task" on page 26-27
Allow this participant to invite other participants (under the Advanced section)	Section 26.6.5.2.4, "Inviting Additional Participants to a Task" on page 26-27

26.6.5.2.1 Assigning Participants to the Single Approver Task

1. Select a method for assigning a user, group, or application role to participate in performing actions on this task.
 - **By name**

- **User Id or Group Id** — Enter a user or group name or click the first icon (**Browse**) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the lookup of user properties, roles, group memberships, and privileges. User information is obtained from the user directory or repository that is configured with the Identity Service. (JAZN, LDAP, OID, or any third party user store). For group assignments, the task must be performed by one of the members in the group. Therefore, the user must first claim it in order to act on it.
- **Application Role** — Enter an application role that contains specific permissions for acting on tasks. For example, assume you have two roles: worker and supervisor. The worker role can only view tasks assigned to the worker group. The supervisor role can view all tasks for their direct reports and reassign tasks. Application roles must be mapped to enterprise roles that are typically in the enterprise LDAP server. During deployment time, you map application roles to enterprise roles in Oracle Enterprise Manager Fusion Middleware Control Console. You can also use Java Platform Security (JPS) policy store APIs to map enterprise roles to application roles.

See Also: [Section 28.1.5, "Identity Service"](#) on page 28-8

- **By expression**

- **Dynamic User XPath or Dynamic Group XPath** — Dynamically assign this task to a user (for example, **jcooper**) or group (for example, **administrators**) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPM Worklist during run time.
- **Dynamic Application Role XPath** — Dynamically assign this task to an application role.

The XPath expressions for specifying assignees must follow these rules:

- They must be based off the task XSD. This includes the payload as defined in the payload section. For example, **/task:task/task:payload/order:orderAssignee** is an example of an XPath expression based of the task XSD.
- The XPath expressions cannot contain BPEL-specific XPath functions such as **bpws:getVariableData()**, and so on because these XPath expressions are not evaluated from the context of a BPEL instance.
- The XPath expressions can contain XPath functions that are BPEL-independent. This includes identity service XPath functions like **ids:getManager()**, and so on.

26.6.5.2.2 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 below a specific amount, no approval is required by their manager.

1. Select the **Specify skip rule** check box. This action displays an icon for accessing the Expression Builder window 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.

See Also: [Section 26.6.5.12, "Advanced Task Routing Using Business Rules"](#) on page 26-42 for instructions on creating dynamic rule conditions

26.6.5.2.3 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.

1. Click the + sign to expand the **Advanced** section shown in [Figure 26–14](#).
2. Select **Limit allocated duration to**.
3. Specify the amount of time.

See Also: [Section 26.6.6, "Escalating, Renewing, or Ending the Task"](#) on page 26-46 for instructions on setting the global escalation and renewal policies in the **Expiration and Escalation Policy** section of the Human Task editor

26.6.5.2.4 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.

1. Click the + sign to expand the **Advanced** section (if not already expanded).
2. Select the **Allow this participant to invite other participants**.

26.6.5.3 Configuring the Group Vote Participant Type

[Figure 26–15](#) and [Figure 26–16](#) display the upper and lower sections of the Group Vote window.

This participant type is used when multiple users, working in parallel, must take action 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 26–15 Add Participant Type — Group Vote (Upper Section of Window)

Type:

Label:
e.g., Approval Manager

Required **consensus** between the participants below: 50

By name By expression

User Id(s)

Enter comma separated names or browser for values.

Group Id(s)

Enter comma separated names or browse for values.

Application Role(s)

Enter comma separated names or browse for values.

Specify skip rule

Skip group vote if condition is satisfied.

Share attachments and comments

Figure 26–16 Add Participant Type — Group Vote (Lower Section of Window)

Majority Voted Outcome

Default Outcome:

Consensus Percentage:
Minimum consensus required to override default outcome

Immediately trigger voted outcome when minimum percentage is met
 Wait until all votes are in before triggering outcome

Advanced

Limit allocated duration to:

Day Hour Minutes
Over all task duration is currently set to 0 days, 0 hours, and 0 minutes.

1. Enter a recognizable label for this participant in the **Label** field. 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 Add Participant Type - Group Vote window are listed in [Table 26–4](#):

Table 26–4 Add Participant Type — Group Vote Window

For This Subsection...	See...
Required consensus between the participants below: 50	Section 26.6.5.3.1, "Assigning Participants to the Group Vote Task" on page 26-29
Specify skip rule	Section 26.6.5.3.2, "Bypassing a Task Participant" on page 26-30
Share attachments and comments	Section 26.6.5.3.3, "Sharing Attachments and Comments with Task Participants" on page 26-30

Table 26–4 (Cont.) Add Participant Type — Group Vote Window

For This Subsection...	See...
Default Outcome Consensus Percentage Immediately trigger voted outcome when minimum percentage is met Wait until all votes are in before triggering outcome	Section 26.6.5.3.4, "Specifying Group Voting Details" on page 26-30
Limit allocated duration to	Section 26.6.5.3.5, "Specifying a Time Limit for Acting on a Task" on page 26-31

26.6.5.3.1 Assigning Participants to the Group Vote Task

- Select a method for assigning a user, group, or application role to participate in this task. A subtask is created for each group vote task participant. The assigned participants must establish a consensus on when a task is considered complete.
 - **By name**
 - **User Id or Group Id** — Enter a user or group name or click the first icon (**Browse**) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the lookup of user properties, roles, group memberships, and privileges. User information is obtained from the user directory or repository that is configured with the Identity Service. (JAZN, LDAP, OID, or any third party user store). For group assignments, the task must be performed by one of the members in the group. Therefore, the user must first claim it in order to act on it.
 - **Application Role** — Enter an application role that contains specific permissions for acting on tasks. For example, assume you have two roles: worker and supervisor. The worker role can only view tasks assigned to the worker group. The supervisor role can view all tasks for their direct reports and reassign tasks. Application roles must be mapped to enterprise roles that are typically in the enterprise LDAP server. During deployment time, you map application roles to enterprise roles in Oracle Enterprise Manager Fusion Middleware Control Console. You can also use JPS policy store APIs to map enterprise roles to application roles.
 - **By expression**
 - **Dynamic User XPath or Dynamic Group XPath** — Dynamically assign this task to a user (for example, `jcooper`) or group (for example, `administrators`) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPM Worklist during run time. For group assignments, the task must be performed by one of the members in the group. Therefore, the user must first claim it in order to act on it. If you want to create a task that must be voted on by each member of the group, use the `orcl:getUsersInGroup` XPath function.
 - **Dynamic Application Role XPath** — Dynamically assign this task to an application role.

See Also: [Section 26.6.5.2.1, "Assigning Participants to the Single Approver Task"](#) on page 26-25 for rules to follow when specifying assignees with XPath expressions

26.6.5.3.2 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 below a specific amount, no approval is required by their manager.

1. Select the **Specify skip rule** check box. This action displays an icon for accessing the Expression Builder window for building a condition. The expression must evaluate to a Boolean value.

See Also: [Section 26.6.5.2.2, "Bypassing a Task Participant"](#) on page 26-26 for an example of a valid XPath expression for skipping a participant

26.6.5.3.3 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 the Oracle BPM Worklist.

1. Select the **Share attachments and comments** check box.

26.6.5.3.4 Specifying Group Voting Details

1. Specify a method for selecting the outcome for the final task. If you select **By Expression** from the lists below, you can dynamically specify the details by clicking the icon to the right of the field to display the Expression Builder window.

- **Default Outcome**

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 window in [Section 26.6.3.2, "Specifying a Task Outcome"](#) on page 26-17 display in this list.

- **Consensus Percentage**

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.

2. 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 already been satisfied.

- **Wait until all votes are in before triggering outcome**

If selected, the workflow waits for all responses before an outcome is initiated.

26.6.5.3.5 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.

1. Click the + sign to expand the **Advanced** section shown in [Figure 26–16](#).
2. Select **Limit allocated duration to**.
3. Specify the amount of time.

See Also: [Section 26.6.6, "Escalating, Renewing, or Ending the Task"](#) on page 26-46 for instructions on setting the global escalation and renewal policies in the **Expiration and Escalation Policy** section of the Human Task editor

26.6.5.4 Configuring the Management Chain Participant Type

[Figure 26–17](#) and [Figure 26–18](#) display the upper and lower sections of the Management Chain window.

This participant type routes tasks for approval to multiple users in a management chain hierarchy. You specify the task participants as a management chain list or a list of users. During the modeling process, you specify the first user to whom the task is assigned and the number of levels. The rest of the users are calculated based on the organization hierarchy with respect to this user.

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 of the managers reject the request or the request expires, the order is rejected.

Figure 26–17 Add Participant Type — Management Chain (Upper Section of Window)

Type:

Label:
e.g., Approval Manager

Requires management chain approval of **one** of the participants below

By name By expression

User Id(s)

Enter comma separated names or browse for values.

Group Id(s)

Enter comma separated names or browse for values.

Application Role(s)

Enter comma separated names or browse for values.

Specify skip rule

Skip remaining management chain participants if condition is satisfied.

Figure 26–18 Add Participant Type — Management Chain (Lower Section of Window)

Number of Approvers:

Maximum Number of Chain Levels Up:
 By Number

Highest Title of Approver:
 By Title

Task will escalate up the chain until the specified maximum level or the highest title has been reached.

Advanced

Limit allocated duration to:
 Day Hour Minutes
Over all task duration is currently set to 0 days, 0 hours, and 0 minutes.

Allow this participant to invite other participants

1. Enter a recognizable label for this participant in the **Label** field. 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 Add Participant Type - Management Chain window are listed in [Table 26–5](#):

Table 26–5 Add Participant Type - Management Chain

For This Subsection...	See...
Requires management chain approval of one of the participants below	Section 26.6.5.4.1, "Assigning Participants to the Management Chain Task" on page 26-32
Specify skip rule	Section 26.6.5.4.2, "Bypassing a Task Participant" on page 26-33
Maximum Number of Chain Levels Up	Section 26.6.5.4.3, "Specifying the Number of Approvers" on page 26-33
Highest Title of Approver	
Limit allocated duration to	Section 26.6.5.4.4, "Specifying a Time Limit for Acting on a Task" on page 26-34
Allow this participant to invite other participants	Section 26.6.5.4.5, "Inviting Additional Participants to a Task" on page 26-34

26.6.5.4.1 Assigning Participants to the Management Chain Task

1. Select a method for assigning a user, group, or application role to participate in this task.
 - **By name**
 - **User Id** or **Group Id** — Enter a user or group name or click the first icon (**Browse**) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the lookup of user properties, roles, group memberships, and privileges. User information is obtained from the user directory or repository that is configured with the Identity Service. (JAZN, LDAP, OID, or any third party user store). For group assignments, the

task must be performed by one of the members in the group. Therefore, the user must first claim it in order to act on it.

- **Application Role** — Enter an application role that contains specific permissions for acting on tasks. For example, assume you have two roles: worker and supervisor. The worker role can only view tasks assigned to the worker group. The supervisor role can view all tasks for their direct reports and reassign tasks. Application roles must be mapped to enterprise roles that are typically in the enterprise LDAP server. During deployment time, you map application roles to enterprise roles in Oracle Enterprise Manager Fusion Middleware Control Console. You can also use JPS policy store APIs to map enterprise roles to application roles.

- **By expression**

- **Dynamic User XPath or Dynamic Group XPath** — Dynamically assign this task to a user (for example, **jcooper**) or group (for example, **administrators**) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPM Worklist during run time.
- **Dynamic Application Role XPath** — Dynamically assign this task to an application role.

See Also: [Section 26.6.5.2.1, "Assigning Participants to the Single Approver Task"](#) on page 26-25 for rules to follow when specifying assignees with XPath expressions

26.6.5.4.2 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 below a specific amount, no approval is required by their manager.

1. Select the **Specify skip rule** check box. This action displays an icon for accessing the Expression Builder window for building a condition. The expression must evaluate to a Boolean value.

See Also: [Section 26.6.5.2.2, "Bypassing a Task Participant"](#) on page 26-26 for an example of a valid XPath expression for skipping a participant

26.6.5.4.3 Specifying the Number of Approvers

You can specify two types of task routing parameters. When both parameters are specified, task routing is determined by both parameters. The routing continues until one of these parameters is satisfied. If you select **By Expression** from the lists below, you can dynamically specify the details by clicking the icon to the right of the field to display the Expression Builder window.

1. Specify the following task routing parameters.

- **Maximum Number of Chain Levels Up**

Enter a value or specify an XPath expression for the number of levels in the management chain to include in this task. For example, if set to 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). This is a mandatory field.

- **Highest Title of Approver**

Select the title or specify an XPath expression of the last (highest) user in the management chain. The title is retrieved from the identity service.

26.6.5.4.4 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.

1. Click the + sign to expand the **Advanced** section shown in [Figure 26–17](#).
2. Select **Limit allocated duration to**.
3. Specify the amount of time.

See Also: [Section 26.6.6, "Escalating, Renewing, or Ending the Task"](#) on page 26-46 for instructions on setting the global escalation and renewal policies in the **Expiration and Escalation Policy** section of the Human Task editor

26.6.5.4.5 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.

1. Click the + sign to expand the **Advanced** section (if not already expanded).
2. Select **Allow this participant to invite other participants**.

Note: For the management chain participant type, the additional participants can be invited only by the last user in the management chain.

26.6.5.5 Configuring the Sequential List of Approvers Participant Type

[Figure 26–19](#) displays the Sequential List of Approvers window.

This 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. This is similar to the management chain participant type, except that with that type, the users are part of an organization hierarchy. For the sequential list of approvers participant type, they can be any list of users or groups.

Figure 26–19 Add Participant Type — Sequential List of Approvers

General

Type: Sequential List Of Approvers

Label:
e.g., Approval Manager

Requires sequential approval of **all** participants below:

By name By expression

Users
 Groups

Id(s):
Enter comma separated names or browse for values.

Specify skip rule

Skip list of single approvers if condition is satisfied.

Advanced

Limit allocated duration to:

Day Hour Minutes

Over all task duration is currently set to {0} days

Allow this participant to invite other participants

1. Enter a recognizable label for this participant in the **Label** field. 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 Add Participant Type - Sequential List of Approvers window are listed in [Table 26–6](#).

Table 26–6 Add Participant Type — Sequential List of Approvers

For This Subsection...	See...
Requires sequential approval of all participants below	Section 26.6.5.5.1, "Assigning Participants to the Sequential List of Approvers Task" on page 26-35
Specify skip rule	Section 26.6.5.5.2, "Bypassing a Task Participant" on page 26-36
Limit allocated duration to	Section 26.6.5.5.3, "Specifying a Time Limit for Acting on a Task" on page 26-36
Allow this participant to invite other participants	Section 26.6.5.5.4, "Inviting Additional Participants to a Task" on page 26-36

26.6.5.5.1 Assigning Participants to the Sequential List of Approvers Task

1. Select a method for assigning a user or group to participate in this task.
 - **By name**

Enter a user or group name or click the first icon (**Browse**) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the lookup of user properties, roles, group memberships, and privileges. User information is obtained from the user directory or repository that is configured with the Identity Service. (JAZN, LDAP, OID, or any third party

user store). For group assignments, the task must be performed by one of the members in the group. Therefore, the user must first claim it in order to act on it.

- **By expression**

Dynamically assign this task to a user (for example, **jcooper**) or group (for example, **administrators**) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPM Worklist during run time.

See Also: [Section 26.6.5.2.1, "Assigning Participants to the Single Approver Task"](#) on page 26-25 for rules to follow when specifying assignees with XPath expressions

26.6.5.5.2 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 below a specific amount, no approval is required by their manager.

1. Select the **Specify skip rule** check box. This action displays an icon for accessing the Expression Builder window for building a condition. The expression must evaluate to a Boolean value.

See Also: [Section 26.6.5.2.2, "Bypassing a Task Participant"](#) on page 26-26 for an example of a valid XPath expression for skipping a participant

26.6.5.5.3 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.

1. Click the + sign to expand the **Advanced** section shown in [Figure 26-19](#).
2. Click **Limit allocated duration to**.
3. Specify the amount of time.

See Also: [Section 26.6.6, "Escalating, Renewing, or Ending the Task"](#) on page 26-46 for instructions on setting the global escalation and renewal policies in the **Expiration and Escalation Policy** section of the Human Task editor

26.6.5.5.4 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.

1. Click the + sign to expand the **Advanced** section (if not already expanded).
2. Select **Allow this participant to invite other participants**.

Note: For the sequential list of approvers participant type, the additional participants can be invited only by the last user in the sequential list.

26.6.5.6 Configuring the FYI Assignee Participant Type

Figure 26–20 displays the FYI Assignee window.

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. FYI assignees 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 26–20 Add Participant Type — FYI Assignee

1. Enter a recognizable label for this participant in the **Label** field. 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 Add Participant Type - FYI Assignee window are listed in [Table 26–7](#):

Table 26–7 Add Participant Type - FYI Assignee

For This Subsection...	See...
Send an FYI copy of this task to all participants below	Section 26.6.5.6.1, "Assigning Participants to the FYI Assignee Task" on page 26-37
Share attachments and comments	Section 26.6.5.6.2, "Sharing Attachments and Comments with Task Participants" on page 26-38

26.6.5.6.1 Assigning Participants to the FYI Assignee Task

1. Select a method for assigning a user, group, or application role to participate in this task.

- **By name**
 - **User Id** or **Group Id** — Enter a user or group name or click the first icon (**Browse**) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the lookup of user properties, roles, group memberships, and privileges. User information is obtained from the user directory or repository that is configured with the Identity Service. (JAZN, LDAP, OID, or any third party user store). For group assignments, the task must be performed by one of the members in the group. Therefore, the user must first claim it in order to act on it.
 - **Application Role** — Enter an application role that contains specific permissions for acting on tasks. For example, assume you have two roles: worker and supervisor. The worker role can only view tasks assigned to the worker group. The supervisor role can view all tasks for their direct reports and reassign tasks. Application roles must be mapped to enterprise roles that are typically in the enterprise LDAP server. During deployment time, you map application roles to enterprise roles in Oracle Enterprise Manager Fusion Middleware Control Console. You can also use JPS policy store APIs to map enterprise roles to application roles.
- **By expression**
 - **Dynamic User XPath** or **Dynamic Group XPath** — Dynamically assign this task to a user (for example, **jcooper**) or group (for example, **administrators**) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPM Worklist during run time.
 - **Dynamic Application Role XPath** — Dynamically assign this task to an application role.

See Also: [Section 26.6.5.2.1, "Assigning Participants to the Single Approver Task"](#) on page 26-25 for rules to follow when specifying assignees with XPath expressions

26.6.5.6.2 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 the Oracle BPM Worklist.

1. Select the **Share attachments and comments** check box.

26.6.5.7 Configuring the External Routing Service Participant Type

[Figure 26–21](#) displays the External Routing Service window.

This participant type enables you to configure an external routing service that dynamically determines the participants in the workflow. If this participant type is specified, all other participant types are ignored. It is assumed that the external routing service provides a list of participant types (single approver, list of approvers, group vote, and so on) at run time to determine the routing of the task.

Use this participant type if you do not want to use any of the built-in workflow participant types or the advanced workflow routing rules to determine task assignees. In this case, all the logic of task assignment is delegated to the external routing service.

Figure 26–21 Add Participant Type — External Routing Service

Type: External Routing Service

Label:
e.g., Approval Manager

ClassName:

Name	Value
<input type="text"/>	<input type="text"/> By name

1. Enter a recognizable label for this participant in the **Label** field. This label must be unique among all the participants in the task definition (for example, Approval Manager, Primary Reviewers, and so on).

26.6.5.7.1 Specifying a Class Name

1. Enter the fully qualified class file name or click the **Browse** icon to select the name (for example, the `org.mycompany.tasks.RoutingService` class name). This class must implement the `oracle.bpel.services.workflow.task.IAssignmentService` interface.
2. Add name and pair value parameters by name or XPath expression that can be passed to the external service.

If You Select...	Then...
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 window.

3. Click the + sign to add additional name and pair value parameters.

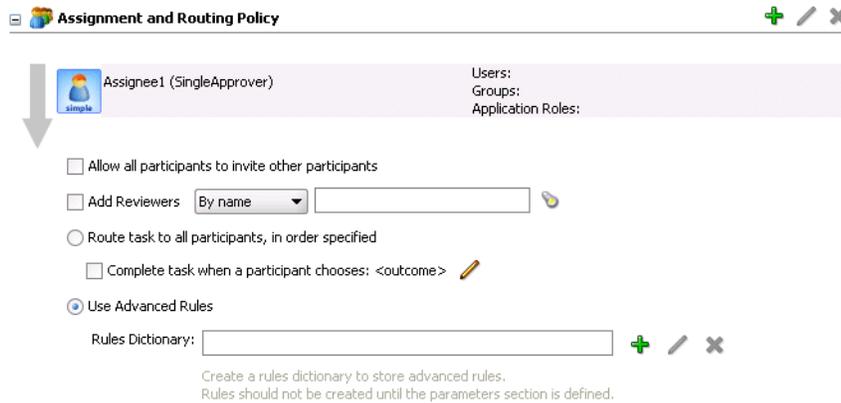
Note: `IAssignmentService` interface details are described in the *Oracle BPEL Process Manager Workflow Services API Reference*

See Also: [Section 28.3.2, "Dynamically Assigning Task Participants with the Assignment Service"](#) on page 28-32 for details about using this interface

26.6.5.8 Allowing All Participants to Invite Other Participants

After you configure a participant type and are returned to the Human Task editor, the **Allow all participants to invite other participants** check box is enabled, as shown in Figure 26–22.

Figure 26–22 Human Task Editor — Assignment and Routing Policy Section



This check box is the equivalent of the Adhoc 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.

1. If you want this task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow, select the **Allow all participants to invite other participants** check box.

26.6.5.9 Adding Reviewers

You can add additional review participants to a task. Participants added through this field can only add comments and attachments to a task. This differs from the task assignees, who can perform actions on the task like reassign, escalate, custom actions (for example, approve and reject), add comments, add attachments, and so on.

You can assign additional reviewers to this task.

1. Select the **Add Reviewers** check box.
2. Select to assign reviewers by name or expression.

If You Select...	Then...
By Name	Enter a user or group name or click the Browse icon to the right of the field to display the Identity Lookup window for selecting a user or group configured through the identity service.
By Expression	Dynamically assign this task to a user (for example, jcooper) or group (for example, administrators) by clicking the icon to the right of the field to display the Expression Builder window.

26.6.5.10 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.

1. Click **Route task to all participants, in order specified** to route tasks to all participants in the specified order.

See Also: [Section 26.6.5.12.1, "Introduction to Advanced Task Routing Using Business Rules"](#) on page 26-42 for details about state machine-based routing

26.6.5.11 Abruptly Completing a Condition

You can specify conditions under which to complete a task early, regardless of the other participants in the workflow.

1. Select the **Complete task when a participant chooses <outcome>** check box.

The Abrupt Completion Details window appears.

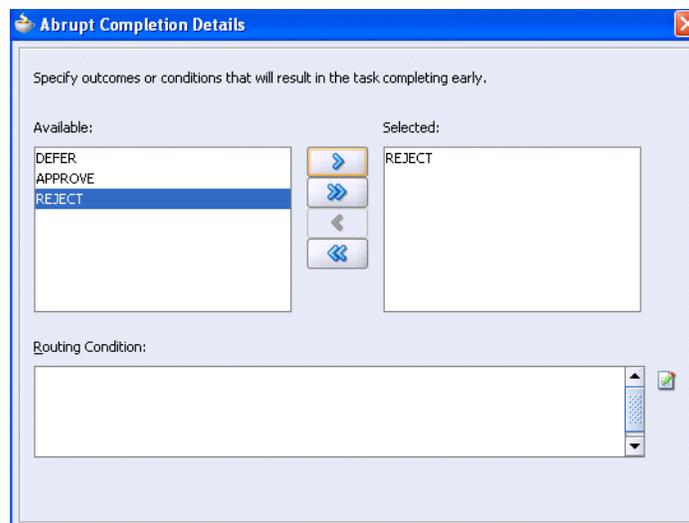
For example, assume an expense report goes to the manager, and then the director. If the first participant (manager) rejects it, you can end the workflow without sending it to the next participant (director).

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.

2. Select appropriate outcomes and click the > button. To select all, click the >> button.



3. Click the icon to the right of the **Routing Condition** field to display the Expression Builder window 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 below a specific amount, no approval is required by their manager.
4. Click **OK** to return to the Human Task editor.

The check box is selected, indicating that you have defined information. You can click the icon to the right of the **Complete task when a participant chooses <outcome>** check box to edit this information.

26.6.5.12 Advanced Task Routing Using Business Rules

The Human Task editor is integrated with the functionality of advanced task routing using business rules.

Use advanced routing rules to create complex workflow routing scenarios. The participant types described in previous sections (such as sequential, management chain, and so on) 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 in conjunction with the Human Task editor.

This section contains the following topics:

- [Section 26.6.5.12.1, "Introduction to Advanced Task Routing Using Business Rules"](#)
- [Section 26.6.5.12.2, "Fact Types"](#)
- [Section 26.6.5.12.3, "Actions"](#)
- [Section 26.6.5.12.4, "Sample Rule Set"](#)
- [Section 26.6.5.12.5, "Creating Advanced Routing Rules"](#)

26.6.5.12.1 Introduction to Advanced Task Routing Using Business Rules You can define state machine routing rules using Oracle Business Rules. This 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 enables you to override the standard task routing slip method described in [Section 26.6.5.10, "Routing Tasks to All Participants in the Specified Order"](#) on page 26-40 and build complex routing behavior into tasks.

26.6.5.12.2 Fact Types 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 26–8](#) describes the fact types asserted by the task service.

Table 26–8 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 enables you to construct tests against payload values and task attribute values.

Table 26–8 (Cont.) Fact Types Asserted By the Task Service

Fact Type	Description
PreviousOutcome	<p>This fact describes the previous task outcome and the assignee who set the outcome. The previous outcome fact contains the following attributes:</p> <ul style="list-style-type: none"> ▪ actualParticipant — The name of the participant who set the task outcome (for example, jstein) ▪ logicalParticipant — The logical name (or label) for the routing slip participant responsible for setting the task outcome (for example, Assignee1) ▪ outcome — The outcome that was set (for example, APPROVE or REJECT) ▪ level — 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. ▪ totalNumberOfApprovals — The total number of users that have set the outcome of the task at this time.
TaskAction	<p>This fact is not intended for writing rule tests against it. Instead, it is updated by rules to indicate how to route the task if the rule is triggered. Rules should not directly update the TaskAction fact. Instead, they should pass the TaskAction fact to one of the RL functions described in Section 26.6.5.12.3, "Actions" on page 26-43. The task service always asserts a TaskAction set with a GO_FORWARD action. This means that the default routing slip behavior (move forward to the next participant in the routing slip) is followed if no rules are triggered by the asserted facts. This eliminates the need for having to write a catch-all rule to implement the default behavior, and makes rule execution more robust.</p>

See Also: *Oracle Fusion Middleware User's Guide for Oracle Business Rules* for details on facts

26.6.5.12.3 Actions In order to instruct the task service on how to route the task, rules can specify one of a number of 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 26–9](#):

Table 26–9 Business Rule Actions

Action	Parameters	Description
GO_FORWARD	TaskAction fact	Goes to the next participant in the routing slip (default behavior).
PUSHBACK	TaskAction fact	Goes back to the previous participant in the routing slip (the participant prior to the one that just set the task outcome).
GOTO	TaskAction fact, String identifying the logical participant name	Goes to a specific participant in the routing slip.

Table 26–9 (Cont.) Business Rule Actions

Action	Parameters	Description
COMPLETE	TaskAction fact	Finishes routing and completes the task. The task is marked as completed, and no further routing is required.
ESCALATE	TaskAction fact	Escalates and reassigns the task according to the task escalation policy (usually to the manager of the current assignee)

26.6.5.12.4 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 desired 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 Rule Author.

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

```

IF

  (for each case where) Task is a Task
    Task.payload.expenseRequest.amount < 100

  and

  (for each case where) PreviousOutcome is a PreviousOutcome
    PreviousOutcome.outcome == "APPROVE"

  and

  (for each case where) TaskAction is a TaskAction

THEN

  call COMPLETE(TaskAction)
    
```

- Push back on the rejected rule

```

IF

  (for each case where) Task is a Task

  and

  (for each case where) PreviousOutcome is a PreviousOutcome
    PreviousOutcome.outcome == "REJECT" and

    PreviousOutcome.logicalAssignee != "Assignee1"

  and

  (for each case where) TaskAction is a TaskAction

THEN

  call PUSHBACK(TaskAction)

```

- Complete the Assignee1 rejected rule

```

IF

  (for each case where) Task is a Task

  and

  (for each case where) PreviousOutcome is a PreviousOutcome
    PreviousOutcome.outcome == "REJECT" and

    PreviousOutcome.logicalAssignee == "Assignee1"

  and

  (for each case where) TaskAction is a TaskAction

THEN

  call COMPLETE(TaskAction)

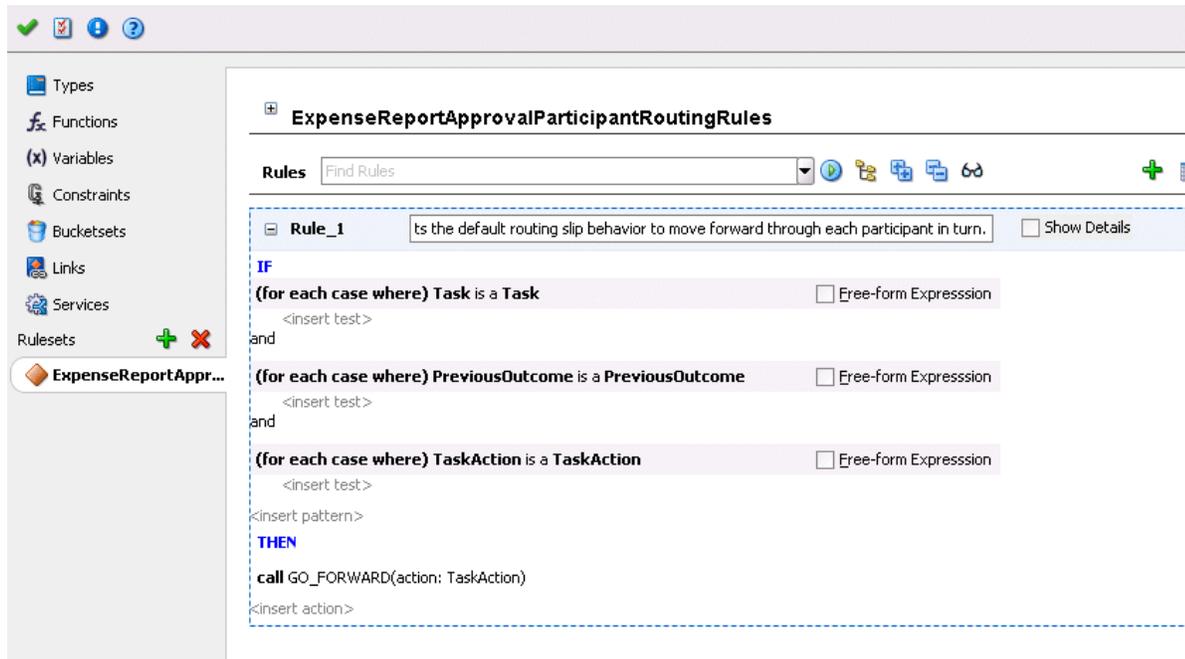
```

See Also: The iterative design sample located in the workflow\workflow-106-IterativeDesign directory of the soa-samples.zip file

26.6.5.12.5 Creating Advanced Routing Rules

1. Click **Use Advanced Rules**.
2. Click **Advanced Routing Rules**.

This starts the Oracle Business Rules Rule Author with a preseeded repository containing all necessary fact definitions. A template rule dictionary is generated that can be used to specify the advanced routing behavior using business rules. The template rule dictionary contains a sample rule that can be copied by right-clicking it. A decision service component is created for the dictionary, and is associated with the task service component.



3. 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.

See Also:

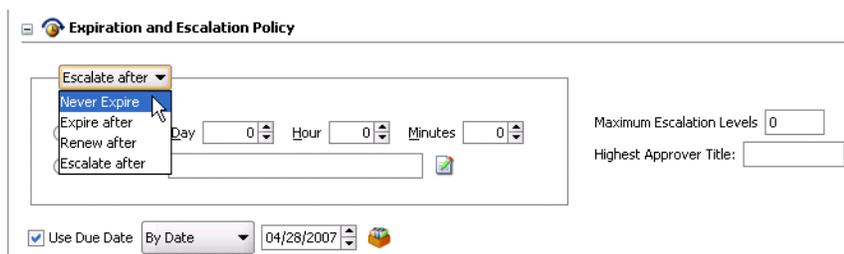
- *Oracle Fusion Middleware User’s Guide for Oracle Business Rules*
- *Oracle Fusion Middleware Language Reference Guide for Oracle Business Rules*

26.6.6 Escalating, Renewing, or Ending the Task

Figure 26–23 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 26–23 Human Task Editor — Expiration and Escalation Policy Section



26.6.6.1 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 3 days to act on the task between them, as shown in [Figure 26–24](#):

Figure 26–24 *Expire After Policy*

The screenshot displays two sections of the Human Task Editor interface:

- Assignment and Routing Policy:**
 - Header: Assignment and Routing Policy (with icons for add, edit, and delete).
 - Assignee: Assignee1 (Group Vote), Required majority: 50 %.
 - Users: LoanAgentGroup, Supervisor.
 - Groups: LoanAgentGroup, Supervisor.
 - Application Roles: (empty).
 - Options:
 - Allow all participants to invite other participants
 - Add Reviewers (By name dropdown, text input, and add icon)
 - Route task to all participants, in order specified
 - Complete task when a participant chooses: <outcome> (with edit icon)
 - Use Advanced Rules
 - Rules Dictionary: (text input, add, edit, and delete icons).
 - Footnote: Create a rules dictionary to store advanced rules. Rules should not be created until the parameters section is defined.
- Expiration and Escalation Policy:**
 - Header: Expiration and Escalation Policy.
 - Expire after dropdown menu.
 - Options:
 - Fixed Duration: Day (input: 3), Hour (input: 0), Minutes (input: 0).
 - By Expression: (text input, add icon).

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 to interpret the participant level expiration for the participants is described below:

- **Management Chain** — Each participant in the management chain gets the same expiration duration. 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 task expires and the escalation and renewal policy is applied.
- **Sequential list of approvers** — Each assignment in the management chain gets the same expiration duration as the one specified in the sequential list of approvers. 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 task expires and the escalation and renewal policy is applied.
- **Group vote**
 - In a group vote 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 is the same as 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.
 - * No expiration duration, otherwise.
- 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.

In the following routing slip sample, participant Loan Agent Group has an expiration duration of 1 day and participant Loan Agent Supervisor does not have any expiration duration on the task, even though an expiration duration is specified at the routing slip level. In this example, the routing slip is treated just as if there were no expiration duration specified at the routing slip level.

```
<routingSlip xmlns="http://xmlns.oracle.com/pcbpel/workflow/routingslip">
  <expirationDuration>PT10D </expirationDuration>

  <participants>

    <participant name="Loan Agent 1" expirationDuration="PT2D">
      <resource isGroup="true" type="STATIC">jcooper</resource>
    </participant>
    <participant name="Loan Agent 2">
      <resource isGroup="true" type="STATIC">jstein</resource>
    </participant>

    <managementChain name="Loan Approval Chain"
      expirationDuration="PT2D">
      <resource isGroup="true" type="STATIC">wfaulk</resource>
      <levels type="STATIC">1</levels>
      <title type="STATIC">Vice President</title>
    </managementChain>

    <participant name="Reviewer">
      <resource isGroup="true" type="STATIC">sfitzger</resource>
    </participant>
  </participants>
</routingSlip>
```

Table 26–10 demonstrates the expiration policy. Note that the management chain in the above example evaluates to two users — wfaulk and cdickens (manager of wfaulk).

Table 26–10 Expiration Policy

Participant	Expiration	Actual Time Taken to Approve
Loan Agent 1 – jcooper	2 days (participant level)	One day
Loan Agent 2 – jstein	9 days (10 – 1 days) (global level)	One day

Table 26–10 (Cont.) Expiration Policy

Participant	Expiration	Actual Time Taken to Approve
Loan Approval Chain – wfaulk (first user in chain)	2 days (participant level)	One day
Loan Approval Chain – cdickens (second user in chain)	2 days (participant level)	One day
Reviewer - sfitzger	6 days (10 – 4 days) (global level)	

- Select an escalation and expiration policy. You can enter a fixed time or a dynamic time by clicking the icon to the right of the **By Expression** field to display the Expression Builder window.
 - Section 26.6.6.2, "Never Expire Policy"
 - Section 26.6.6.3, "Expire After Policy"
 - Section 26.6.6.4, "Renew After Policy"
 - Section 26.6.6.5, "Escalate After Policy"

26.6.6.2 Never Expire Policy

You can specify for a task to never expire.

- Select **Never Expire** from the list shown in [Figure 26–23](#) on page 26-46.

26.6.6.3 Expire After Policy

You can specify for a task to expire.

- Select **Expire after** from the list shown in [Figure 26–23](#) on page 26-46.
- Specify the maximum time period for the task to remain open.

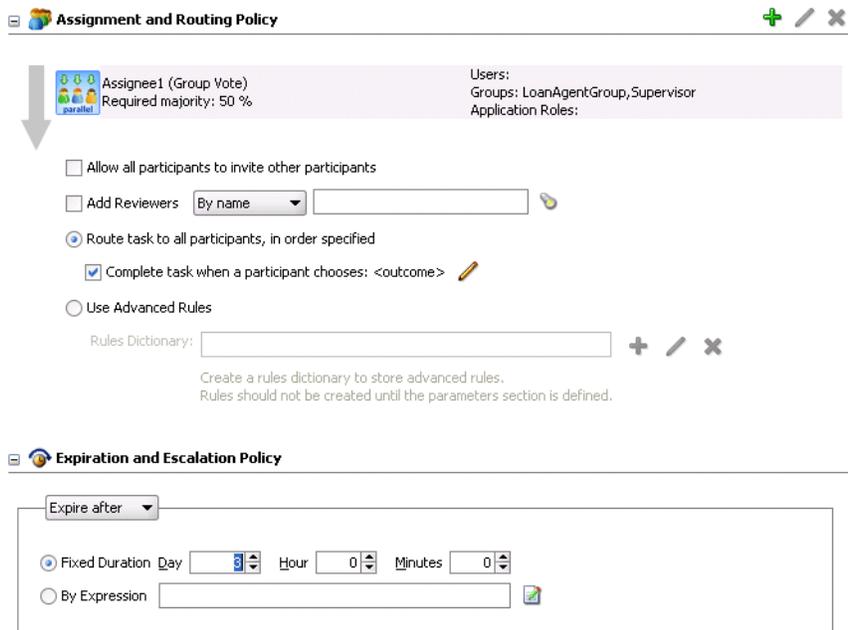
When the task expires, either the escalation policy or the renewal policy at the routing slip level is applied. If neither is specified, the task expires. The expiration policy at the routing slip level is common to all the participants.

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 26–25](#) indicates that the task expires in 3 days.

Figure 26–25 Expire After Policy



26.6.6.4 Renew After Policy

You can extend the expiration period when the user does not respond within the allotted time.

1. Select **Renew after** from the list shown in [Figure 26–23](#) on page 26-46.
2. Specify the maximum number of times to continue renewing this task.

The renewal policy specifies the number of times the task can be renewed on expiration and the renewal duration. In [Figure 26–26](#), when the task expires, it is renewed at most 3 times. It does not matter if the task expired at the **LoanAgentGroup** participant or the **Supervisor** participant.

Figure 26–26 Renew After Policy

Assignment and Routing Policy

Assignee1 (Group Vote)
Required majority: 50 %

Users:
Groups: LoanAgentGroup, Supervisor
Application Roles:

Allow all participants to invite other participants

Add Reviewers By name

Route task to all participants, in order specified

Complete task when a participant chooses: <outcome>

Use Advanced Rules

Rules Dictionary: + / x

Create a rules dictionary to store advanced rules.
Rules should not be created until the parameters section is defined.

Expiration and Escalation Policy

Renew after ▼

Fixed Duration Day Hour Minutes Maximum Renewals:

By Expression

26.6.6.5 Escalate After Policy

You can escalate the task (for example, to the user’s manager) if the user does not respond within the allotted time.

1. Select **Escalate after** from the list shown in [Figure 26–23](#) on page 26-46.
2. Specify the following additional values:
 - **Maximum Escalation Levels**
Number of management levels to which to escalate the task
 - **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.

The escalation policy specifies the number of times the task can be escalated on expiration and the renewal duration. In [Figure 26–27](#), when the task expires, it is escalated at most 3 times. It does not matter if the task expired at the **LoanAgentGroup** participant or the **Supervisor** participant.

Figure 26–27 Escalate After Policy

The screenshot displays two configuration panels in the Human Task Editor. The top panel, titled "Assignment and Routing Policy", includes a header with a plus, edit, and close icon. Below the header, it shows "Assignee1 (Group Vote)" with a "Required majority: 50 %" and "parallel" status. To the right, it lists "Users:", "Groups: LoanAgentGroup, Supervisor", and "Application Roles:". The main area contains several options: "Allow all participants to invite other participants" (unchecked), "Add Reviewers" (with a "By name" dropdown and an input field), "Route task to all participants, in order specified" (selected), "Complete task when a participant chooses: <outcome>" (unchecked), and "Use Advanced Rules" (unselected). Below these is a "Rules Dictionary:" field with a plus, edit, and close icon, and a note: "Create a rules dictionary to store advanced rules. Rules should not be created until the parameters section is defined." The bottom panel, titled "Expiration and Escalation Policy", has a header with a plus, edit, and close icon. It features an "Escalate after" dropdown menu. Below this, there are two radio buttons: "Fixed Duration" (selected) and "By Expression". The "Fixed Duration" section includes "Day" (3), "Hour" (0), and "Minutes" (0) spinners, along with "Maximum Escalation Levels" (0) and "Highest Approver Title" (Director) dropdowns.

26.6.6.6 Specifying a Due Date

You can enter a due date for a task, as shown in [Figure 26–23](#) on page 26-46. A task is considered overdue once 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 the 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.

1. Select **Use Due Date**.
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 task dialog) 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 run-time 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.

See Also: [Section 33.3.4, "How to Create a To-Do Task"](#) on page 33-14

26.6.7 Specifying Participant Notification Preferences

Figure 26–28 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 e-mail, 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 26–28 Human Task Editor — Notification Settings Section

Task Status	Recipient	Notification Header
Assign	Assignees	
Complete	Initiator	
Error	Owner	

Remind once: Day Hour Minutes Before Expiration

Make notifications secure (exclude details)
 Make email messages actionable
 Send task attachments with email notifications

Custom Notification Headers

Name	Value

1. Click the + sign to expand the **Notification Settings** section (displays as shown in Figure 26–28).

Instructions for configuring the following subsections of the **Notification Settings** section are listed in Table 26–11.

Table 26–11 Human Task Editor — Notification Settings Section

For This Subsection...	See...
Task Status	Section 26.6.7.1, "Notifying Recipients of Changes to Task Status" on page 26-54
Recipient	
Notification Header	Section 26.6.7.2, "Editing the Notification Message" on page 26-55
Reminders	Section 26.6.7.3, "Setting Up Reminders" on page 26-55
Make notifications secure (exclude details)	Section 26.6.7.4, "Securing Notifications, Making Messages Actionable, and Sending Attachments" on page 26-56
Make e-mail messages actionable	
Send task attachments with email notifications	

See Also: [Section 28.2, "Notifications from Human Workflow"](#) on page 28-20

26.6.7.1 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.

1. Click a type in the **Task Status** column to display the complete list of task types:
 - **Assign**—when the task is assigned to users or a group. This action captures the following actions:
 - Task is assigned to a user
 - Task is assigned to a new user in a sequential list of approvers workflow
 - Task is renewed
 - Task is delegated
 - Task is reassigned
 - Task is escalated
 - Information for a task is submitted
 - **Complete**
 - **Error**
 - **Expire**
 - **Request Info**
 - **Update Outcome**
 - **Suspend**
 - **Resume**
 - **Withdraw**
 - **Update**
 - Task payload is updated
 - Task is updated
 - Comments are added
 - Attachments are added and updated
 - **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. Click an entry in the **Recipient** column 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 approved the task so far. This applies in a sequential list of approvers participant type where multiple users have approved the task and a notification must be sent to all of them.

- **Owner**—the task owner

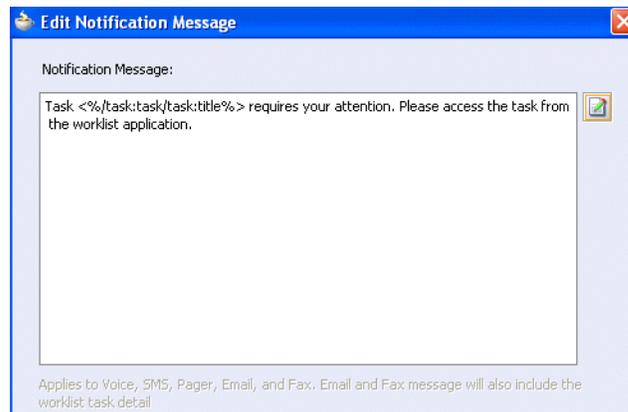
See Also: [Section 28.2.1, "Configuring the Notification Channel"](#) on page 28-21

26.6.7.2 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.

1. Click the icon in the **Notification Header** column to modify the default notification message.

The Edit Notification Message window appears.



This message applies to all the supported notification channels: e-mail, voice, instant message, and SMS. E-mail 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.

See Also: [Section 28.2, "Notifications from Human Workflow"](#) on page 28-20 for notification preference details

26.6.7.3 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.

1. Select the number of reminders to send from the **Remind** list.
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.

Notification Settings

Task Status	Recipient	Notification Header
Assign	Assignees	
Complete	Initiator	
Error	Owner	

Remind three times ▾ Day Hour Minutes After Assignment ▾

See Also: [Section 28.2.12, "Sending Reminders"](#) on page 28-28

26.6.7.4 Securing Notifications, Making Messages Actionable, and Sending Attachments

You can perform additional notification tasks in this section.

1. Select the corresponding check box for functionality you want to use.

Field	Value
Make notifications secure (exclude details)	<ol style="list-style-type: none"> 1. Select to make the notification message secure. If selected, a default notification message is used. There are no HTML worklist task details, attachments, or actionable links in the e-mail. Only the task number is in the message. See Also: Section 28.2.9, "Sending Secure Notifications" on page 28-28
Make e-mail messages actionable	<ol style="list-style-type: none"> 1. Select to make e-mail notifications actionable. This enables you to perform task actions through e-mail. See Also: Section 28.2.4, "Sending Actionable Messages" on page 28-24 for additional configuration details and <i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite</i> for instructions on configuring outbound and inbound e-mails

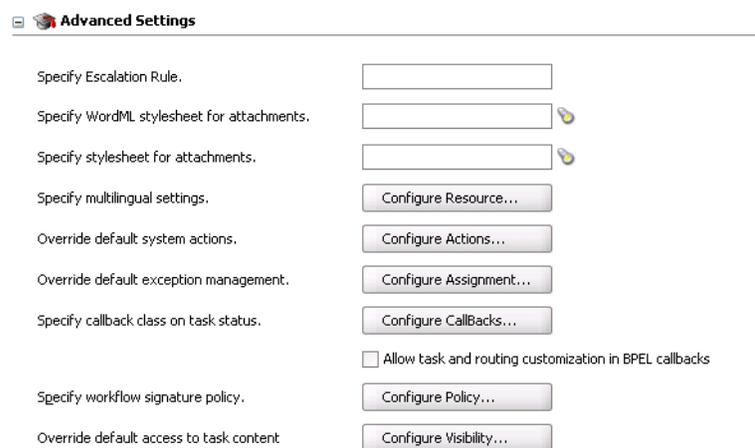
Field	Value
Send task attachments with e-mail notifications	<ol style="list-style-type: none"> Select to send task attachments with the notification message. If you want to customize the notification headers, select Custom Notification Headers. Custom notification headers are used to specify name and 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 maybe Name to Priority and value to High. Users can then use specify delivery preferences in the worklist application. These preferences can be based on the contents of the notification. Note that the rule-based notification service is <i>only</i> used to identify the preferred notification channel to use. The address for the preferred channel is still obtained from the identity service. Add name and pair value parameters by name or XPath expression. See Also: "Sending Inbound and Outbound Attachments" on page 28-27, "Custom Notification Headers" on page 28-29, Chapter 40, "User Messaging Preferences", and preferences details in the Using Oracle BPM Worklist chapter.

26.6.8 Specifying 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, custom task actions and error messages, and callback classes.

[Figure 26–29](#) shows the advanced settings section of the Human Task editor.

Figure 26–29 Human Task Editor — Advanced Settings Section



[Table 26–12](#) describes the sections available.

Table 26–12 Advanced Settings Sections

Section	See...
Specify escalation rule	Section 26.6.8.1, "Specifying Escalation Rules" on page 26-58
Specify WordML Stylesheet for attachments	Section 26.6.8.2, "Specifying WordML Style Sheets for Attachments" on page 26-59
Specify stylesheet for attachments	Section 26.6.8.3, "Specifying Style Sheets for Attachments" on page 26-59
Specify multilingual settings	Section 26.6.8.4, "Specifying Multilingual Settings" on page 26-59
Override default exception management	Section 26.6.8.5, "Overriding Default Exception Management" on page 26-60
Specify callback class on task status	Section 26.6.8.6, "Specifying Callback Classes on Task Status" on page 26-61
Specify workflow signature policy	Section 26.6.8.8, "Specifying a Workflow Signature Policy" on page 26-62
Allow task and routing customization in BPEL callbacks	Section 26.6.8.7, "Allowing Task and Routing Customization in BPEL Callbacks" on page 26-62
Override default access to task content	Section 26.6.8.9, "Specifying Access Rules on Task Content" on page 26-63

26.6.8.1 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 the steps below.

1. Implement interface `oracle.bpel.services.workflow.assignment.dynamic.IDynamicTaskEscalationFunction`. This implementation has to be available in the classpath for the server.
2. Change the file `SOA_Oracle_Home\bpel\system\services\config\wf-dynamic-assign-cfg.xml` to add a new function:

```
<dynamicAssignmentFunctions>

    <function name="Department_supervisor"
    classpath="oracle.bpel.services.workflow.assignment.dynamic.patterns.
    DepartmentSupervisor">
    </function>

</dynamicAssignmentFunctions>
```

Note: For 11g Release 1, this parameter must be configured from the Task Service page in Oracle Enterprise Manager Grid Control Console. Access this page by selecting **Administration > Human Workflow** from the **SOA Infrastructure** menu, then selecting the **Task Service** tab.

3. Enter the function name as defined in the `wf-dynamic-assign-cfg.xml` file for the escalation rule in the **Specify Escalation Rule** field.

See Also: [Section 28.3.3, "Custom Escalation Function"](#) on page 28-36

26.6.8.2 Specifying WordML Style Sheets for Attachments

This option allows dynamic creation of Microsoft Word documents for the purpose of sending them as e-mail attachments using a WordML XSLT stylesheet. The XSLT stylesheet is applied on the task document.

1. Click the **Browse** icon to select a WordML style sheet as an attachment.

See Also: *Oracle Application Server Developer's Guide for Microsoft Office Interoperability* for specific details

26.6.8.3 Specifying Style Sheets for Attachments

This option allows creation of e-mail attachments using an XSLT stylesheet. The XSLT stylesheet is applied on the task document.

1. Click the **Browse** icon to select a stylesheet as an attachment.

26.6.8.4 Specifying Multilingual Settings

You can specify resource bundles for displaying task details in different languages in the Oracle BPM Worklist. Resource bundles are supported for the following task details.

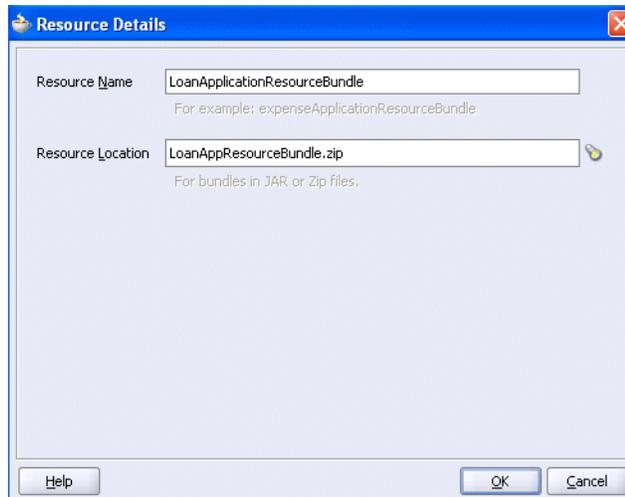
- Displaying the value for task outcomes in plain text or with the message (key) format
- Displaying the XML element and attributes names in the payload display of the Oracle BPM Worklist. The key name in the resource bundle must be the same as the name of the XML element and attributes for internationalization of XML element names in the Oracle BPM Worklist.
- Making e-mail notification messages available in different languages. At run time, 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

1. Click **Configure Resource**.

The Resource Details window appears.



2. Enter the name of the resource used in the resource bundle.
3. Click the **Browse** icon to select the JAR or ZIP resource bundle file to use. The resource bundle is part of your system archive (SAR) file.
4. Click **OK** to return to the Human Task editor.

See Also: [Section 28.2.3, "Configuring Notification Messages in Different Languages"](#) on page 28-23

26.6.8.5 Overriding Default Exception Management

Tasks can error due to incorrect assignments. Incorrect assignments can occur for any of the following reasons:

- Invalid assignees — The task assignee user or group is not a valid user in the system.
- Invalid dynamic assignment — When assignees are specified to be dynamic, the dynamic XPath expression is not evaluated.

In the above cases, the task is marked as errored. By default, the life cycle of the task is completed with that action.

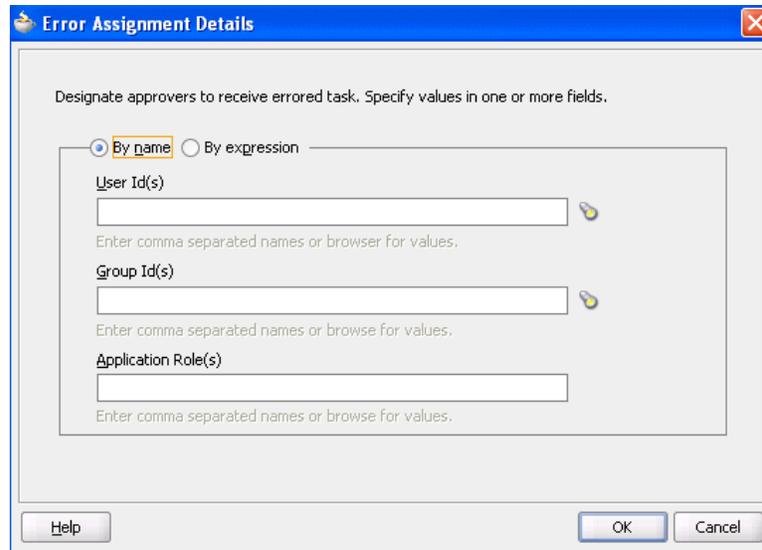
During modeling of workflow tasks, you can specify error assignees for the workflow. If error assignees are specified, they are evaluated and the task is assigned to them. The error assignee can perform one of the following actions:

- Adhoc route — route the task to the actual users assigned to the task. Adhoc route 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 errored.

This window enables you to specify the users or groups to whom the task is assigned if an error in assignment has occurred.

1. Click **Configure Assignment**.
2. Select the error assignees.



Error Assignment Details

Designate approvers to receive errored task. Specify values in one or more fields.

By name By expression

User Id(s)
 
 Enter comma separated names or browser for values.

Group Id(s)
 
 Enter comma separated names or browse for values.

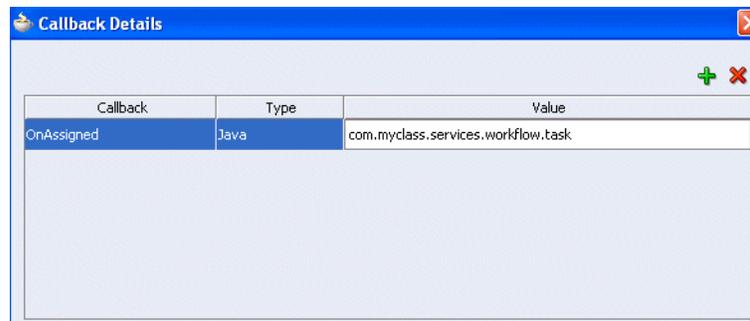
Application Role(s)

 Enter comma separated names or browse for values.

26.6.8.6 Specifying Callback Classes on Task Status

You can register callbacks for the workflow service to call during the life cycle of a task. The callback class has to implement the interface `oracle.bpel.services.workflow.task.IRoutingSlipCallback`. Make the callback class available in the classpath of the server.

1. Click **Configure Callbacks**.
2. Click the + sign to add a callback to the table. A callback named **OnAssigned** is automatically added to the **Callback** column.



Callback Details

Callback	Type	Value
OnAssigned	Java	com.myclass.services.workflow.task

3. Click **OnAssigned** to display a list of additional callback values to select for this column.

The following callbacks are available:

- **OnCompleted** — This callback is invoked when the task is completed, expired, withdrawn, or errored.
- **OnAssigned** — 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
- **OnUpdated** — 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 a task due to request for information, submit information, escalation, reassign, and so on.

- **OnSubtaskUpdated** — This callback is invoked for any update to a subtask.
4. Click **Java** in the **Type** column to display a list of additional values for this column.
 5. Click the empty field in the **Value** column to enter a value. The value is the complete class name of the Java class that implements `oracle.bpel.services.workflow.task.IRoutingSlipCallback`.
 6. Click **OK**.

26.6.8.7 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. In addition, you must enable this in the human task activity in the BPEL process.

The **Allow task and routing customization in BPEL callbacks** check box must be selected if you select the check box of the same name on the Human Task - Advanced tab shown in [Figure 26–36](#) on page 26-77. Selecting both check boxes updates the metadata for callbacks.

See Also: [Section 26.7.4.5, "Using Task and Routing Customizations in BPEL Callbacks"](#) on page 26-79 for details on using callbacks

26.6.8.8 Specifying a Workflow Signature Policy

Digital signatures provide a mechanism for the nonrepudiation of digitally-signed human tasks. This ability to ensure that the original signed message arrived means that the sender cannot repudiate it later.

1. 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 must specify a signature before sending tasks to the next participant. Participants must re-enter 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. This enables the recipient to be sure of the sender's identity.
 - **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 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. The CA names and CA CRL and URLs of the issuing authorities must be configured separately.
3. Click **OK**.

See Also: [Section 28.1.10, "Digital Signatures and the Evidence Store Service"](#) on page 28-16

26.6.8.9 Specifying Access Rules 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.

This section contains the following topics:

- [Section 26.6.8.9.1, "Introduction to Access Rules"](#)
- [Section 26.6.8.9.2, "Specifying Task Content Access Rules"](#)
- [Section 26.6.8.9.3, "Specifying Task Actions Access Rules"](#)

Note: Task contents access rules and task actions access rules exist independently of one another.

26.6.8.9.1 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 means *only* assignees and nobody else 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.  
public Map<String, IPrivilege> getTaskVisibilityRules(IWorkflowContext context,  
String taskId)  
throws TaskMetadataServiceException;
```

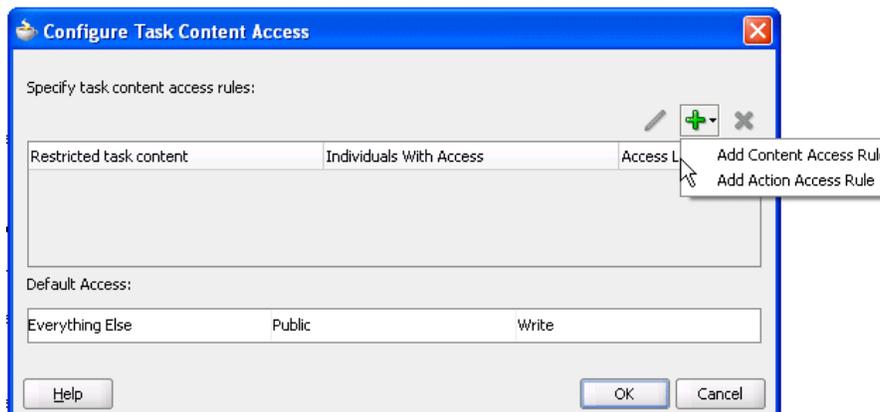
See Also: *Oracle BPEL Process Manager Workflow Services API Reference* for additional details about this method

26.6.8.9.2 Specifying Task Content Access Rules You can specify the access levels that specific users (such as the approval manager) have to act on task attributes (such as a payload).

1. Click **Configure Visibility**.

The Configure Task Content Access window appears.

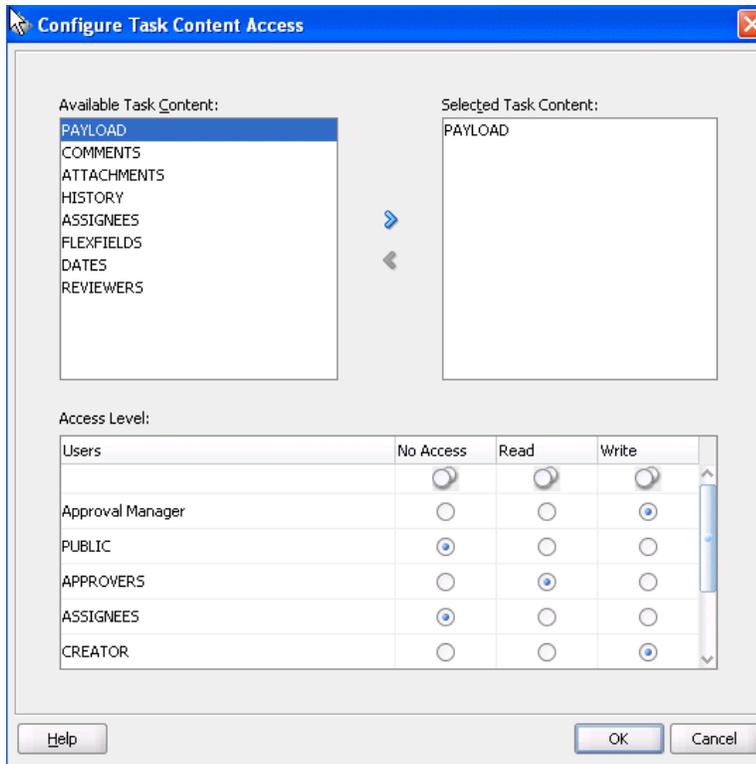
2. Click the **Create** icon and select **Add Content Access Rule**.



The Configure Task Content Access window appears.

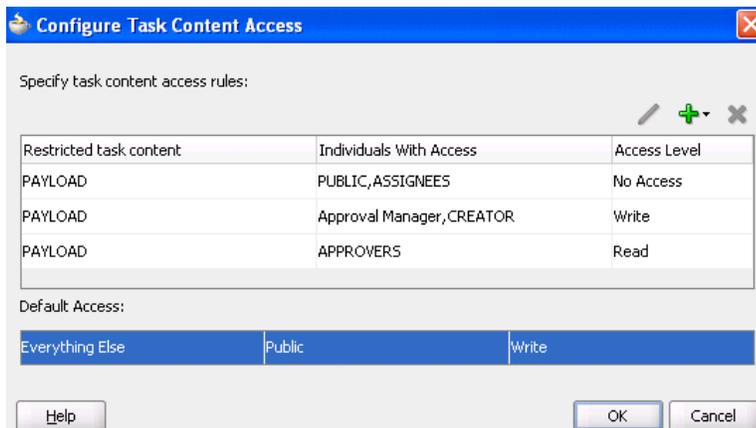
3. Select the task attribute on which to specify access levels:
 - **PAYLOAD**
 - **COMMENTS**

- **ATTACHMENTS**
 - **HISTORY**
 - **ASSIGNEES** (covers assignees, assignee users, and assignee groups)
 - **FLEXFIELDS**
 - **DATES** (Covers the start date, end date, assigned date, created date, expiration date and updated date of comments, task, and so on)
 - **REVIEWERS**
 - Payload message attributes
4. Select the participants and their privileges (no access, read access, or write access) for acting upon the selected task attribute. Participants that you do not select have complete access.
- **Approval Manager**
 - **PUBLIC**
 - **APPROVERS**
 - **ASSIGNEES**
 - **CREATOR**
 - **OWNER**
 - **REVIEWERS**
 - **ADMIN**
 - Individual participants as specified in the routing slip of the task definition
- For example, if you provide the **Approval Manager** and **CREATOR** users with write access, the **APPROVERS** user with read access, and the **PUBLIC** and **ASSIGNEES** users with no access on the **PAYLOAD** task attribute, the window appears as follows:



5. Click **OK**.

The Configure Task Content Access window displays details about your selections.



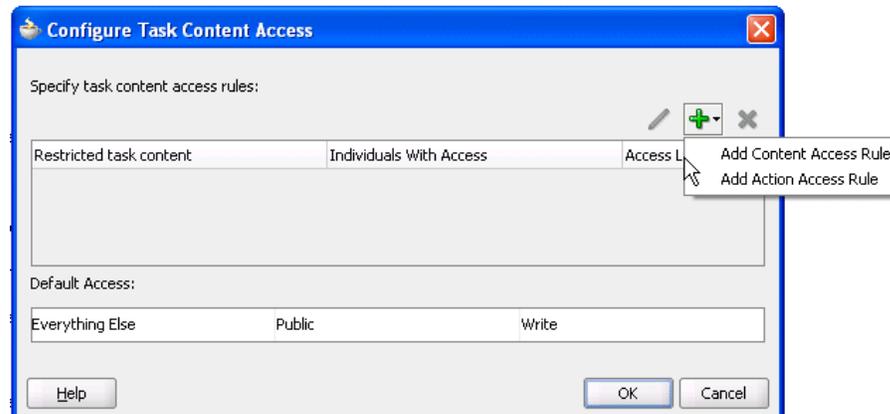
6. Click **OK**.

26.6.8.9.3 Specifying Task Actions Access Rules You can specify the actions that specific users (such as the approval manager) have to act on task attributes (such as a **payload**) that you specified in the Configure Task Content Access window.

1. Click **Configure Visibility**.

The Configure Task Content Access window appears.

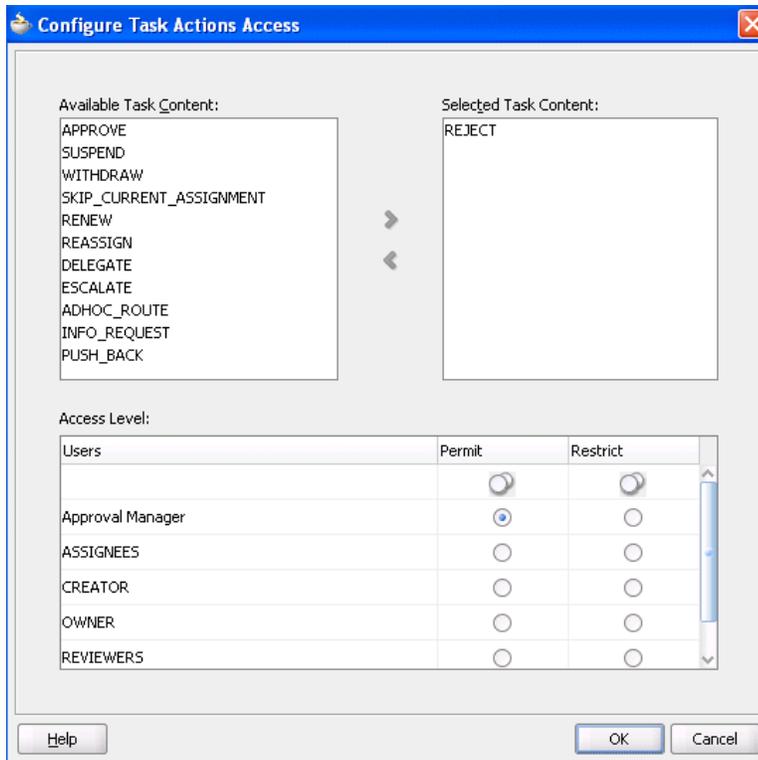
2. Click the **Create** icon and select **Add Action Access Rule**.



The Configure Task Actions Access window appears.

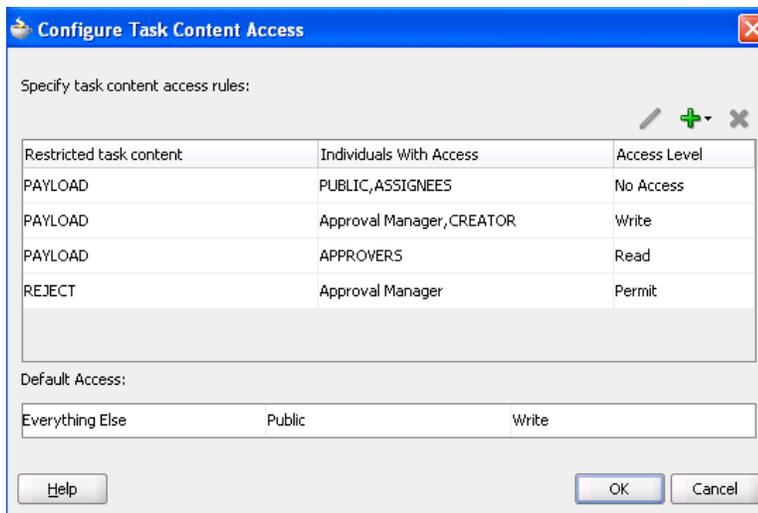
3. Select the actions that can be performed on task attributes.
 - APPROVE
 - SUSPEND
 - WITHDRAW
 - SKIP_CURRENT_ASSIGNMENT
 - RENEW
 - REASSIGN
 - DELEGATE
 - ESCALATE
 - ADHOC_ROUTE
 - INFO_REQUEST
 - PUSH_BACK
4. Select the participants and whether they are permitted or restricted to perform the selected actions:
 - Approval Manager
 - ASSIGNEES
 - CREATOR
 - OWNER
 - REVIEWERS
 - ADMIN

For example, if you permit the approval manager to have the **REJECT** action, the window appears as follows:



5. Click **OK**.

The Configure Task Content Access window appears.



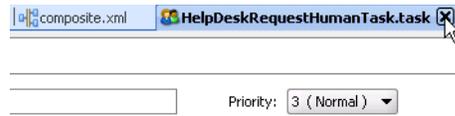
26.6.9 Specifying Annotations

Annotations are used to label different attributes of the task definition. This functionality is used with Oracle Business Process Analysis where 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 modelling process.

26.6.10 Exiting the Human Task Editor and Saving 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**.

1. Select **Save** from the **File** main menu or click the **X** sign to close the `.task` metadata task configuration file.



2. If you click the **X** sign, select **Yes** when prompted to save your changes.

26.7 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. The Task Service exposes the operations required to act on a task.

See Also: [Section 26.6, "Creating the Human Task Definition with the Human Task Editor"](#) on page 26-12 for instructions on creating a human task

26.7.1 Associating a Human Task with a BPEL Process

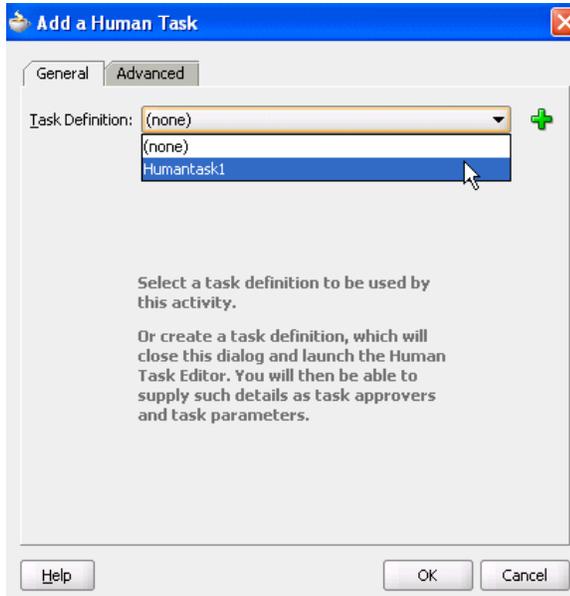
Note: You must already have created a human task service component in the SOA Composite Editor before you can associate it with a BPEL process. If you have not, you are unable to provide a value in the **Task Definition** field of the Add Human Task window.

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. Select **BPEL** from the **Component Palette**.
4. Expand **BPEL Activities**.
5. Drag and drop a new **Human Task** activity into the BPEL process.

The Add a Human Task window appears.

Note: When you first drag and drop this activity into Oracle JDeveloper, the window is named *Add a Human Task*. After you finish specifying details on this window and click **OK**, the name of this window changes to simply *Human Task*.

6. Select the human task from the **Task Definition** list.



The .task file of the human task service component is associated with the BPEL process.

7. See the following sections to configure the human task activity:
 - [Section 26.7.2, "Defining the Human Task Activity Title, Initiator, Priority, and Parameter Variables"](#)
 - [Section 26.7.4, "Defining the Human Task Activity Advanced Features"](#)

Note: After you complete association of your human task activity with a BPEL process and close the Add a Human Task window, you can always re-access this window by double-clicking the human task activity in Oracle JDeveloper.

26.7.2 Defining the Human Task Activity Title, Initiator, Priority, and Parameter Variables

Figure 26–30 shows the **General** tab that displays after you select the human task.

Figure 26–30 Human Task — General Tab (After Selection)

The **General** tab of the Human Task activity enables you to perform the tasks shown in [Table 26–13](#):

Table 26–13 Human Task - General Tab

For this Field...	See...
Task Title	Section 26.7.2.1, "Specifying the Task Title" on page 26-71
Initiator	Section 26.7.2.2, "Specifying the Task Initiator and Task Priority" on page 26-72
Priority	
Task Parameters	Section 26.7.2.3, "Specifying Task Parameters" on page 26-72

26.7.2.1 Specifying the Task Title

1. Enter the task title in the **Task Title** field through one of the following methods. 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 26.6.3.1, "Specifying a Task Title and Priority"](#) on page 26-16. The title displays the task in the Oracle BPM Worklist during run time.
 - Enter the title manually.
 - Click the icon to the right of the field to display the Expression Builder window 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 window.

See Also: [Section 26.8.3.5, "Associating the Human Task and BPEL Process Service Components"](#) on page 26-88 for an example of specifying the task title name

26.7.2.2 Specifying the Task Initiator and Task Priority

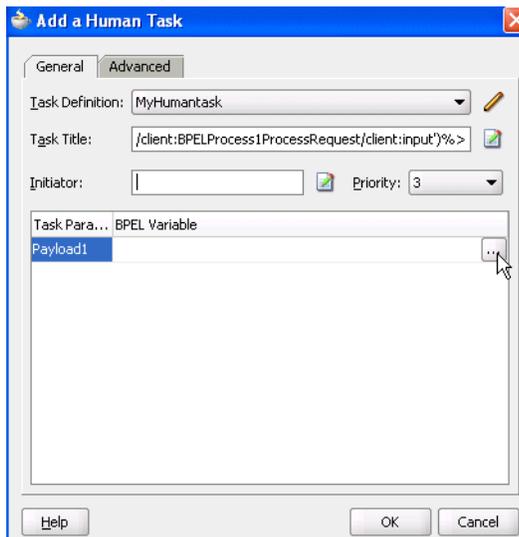
You can specify a task initiator. The initiator is the user who initiates a task. The initiator can view their created tasks from the Oracle BPM Worklist and perform specific tasks defined in the System Action Details window, such as withdrawing or suspending a task.

1. Enter the initiator (for example, `jcooper`) or click the icon to the right of the **Initiator** field to display the Expression Builder window 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 window. The initiator defaults to `bpeladmin` if a task owner is also not specified.
2. Select a priority value between **1** (the highest) and **5** from the **Priority** list. This field is provided for user reference and does not make this task a higher priority during run time. The priority can be used to sort tasks in the Oracle BPM Worklist. This priority value overrides the priority value you select in the **Priority** list of the Human Task editor.

See Also: [Section 26.6.3.1, "Specifying a Task Title and Priority"](#) on page 26-16 for instructions on specifying the priority in the Human Task editor

26.7.2.3 Specifying Task Parameters

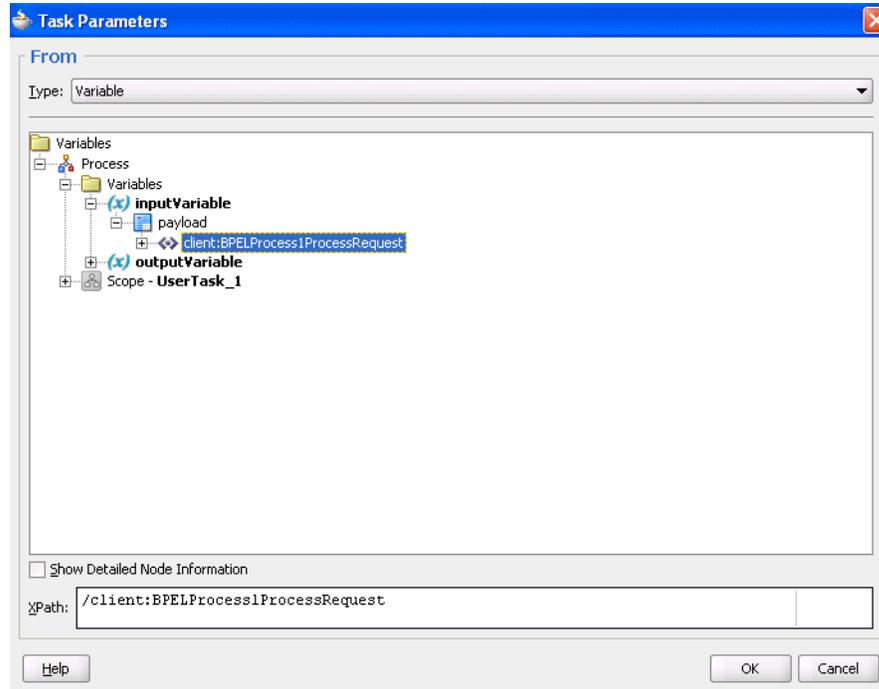
The task parameter table displays a list of task parameters after you complete the **Task Title** and **Initiator** fields.



1. Double-click the **dots** in the **BPEL Variable** column to map the task parameter to the BPEL variable. You must map only the task parameters that carry input data. For output data that is filled in from the worklist, you do not need to map the corresponding variables.

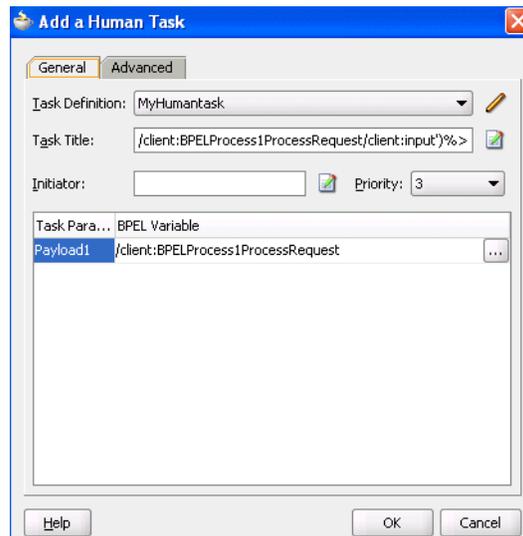
The Task Parameters window appears.

2. Expand the **Variables** navigation tree and select the appropriate task variable.



3. Click **OK**.

The Human Task window appears as follows.



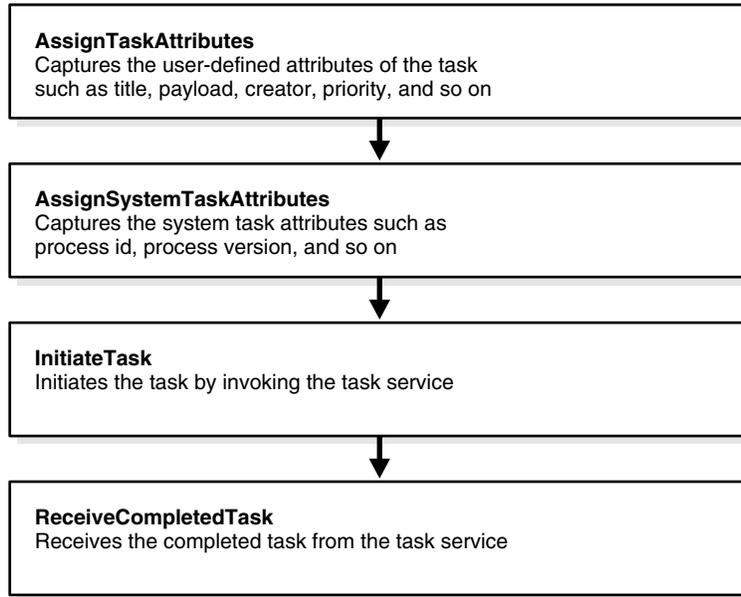
4. Click **OK**.
5. If you want to define advanced features for the human task activity, click the **Advanced** tab and go to section [Section 26.7.4, "Defining the Human Task Activity Advanced Features"](#) on page 26-77. Otherwise, click **OK** to close the Human Task window.

26.7.3 Viewing the Generated Human Task Activity

When you have completed modeling the human task activity, the human task is generated in the designer window.

Figure 26–31 shows how a workflow interaction is modeled in Oracle BPEL Process Manager. Figure 26–31 also illustrates the interaction when no BPEL callbacks are modeled. In this case, once 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 26–31 Workflow Interaction Modeling

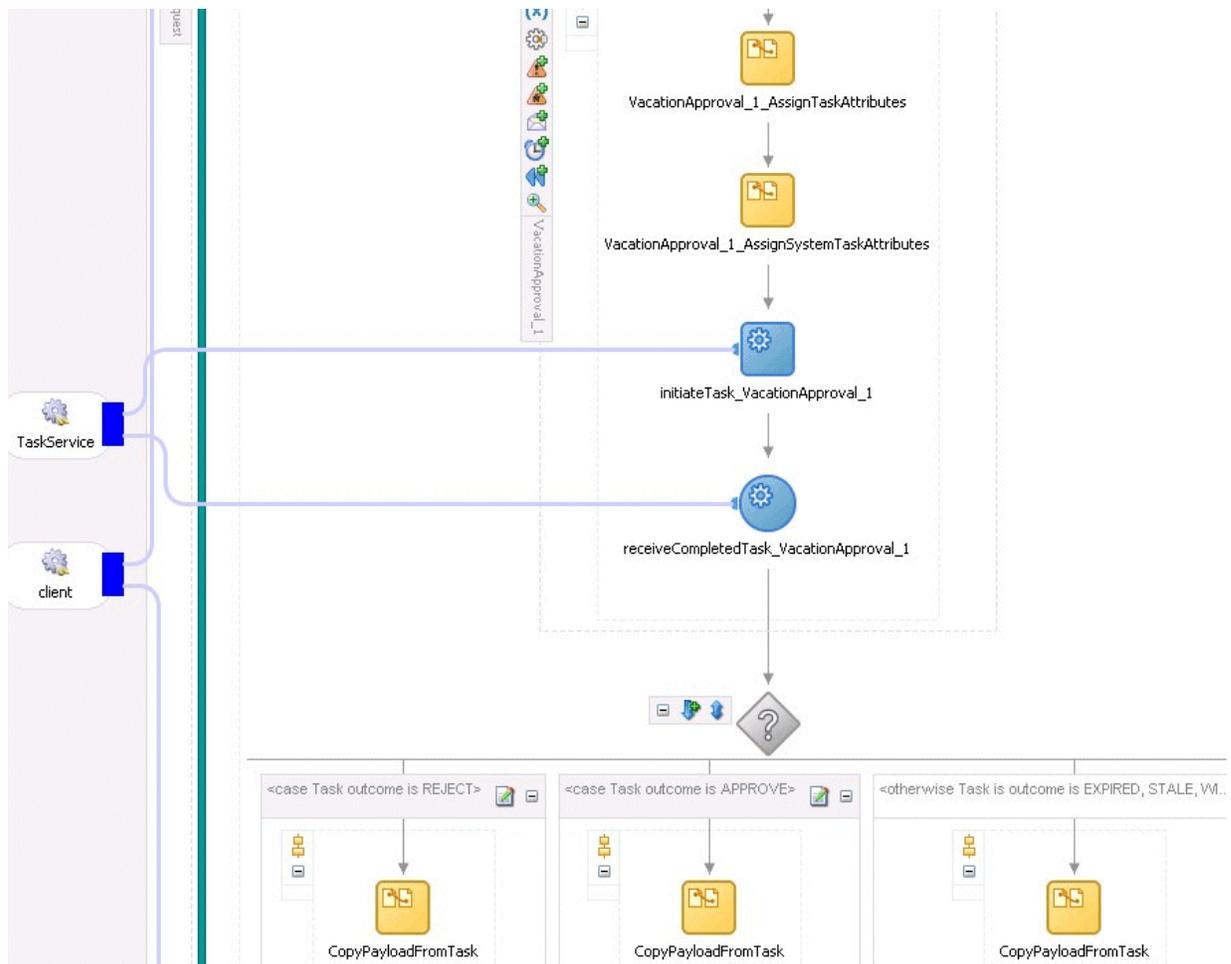


Click the + sign next to the human task activity in Oracle JDeveloper to display its contents, as shown in Figure 26–32.

Figure 26–32 Expanding the Human Task Activity



Figure 26–33 shows the workflow interaction in Oracle JDeveloper.

Figure 26–33 Workflow Interaction Modeling in Oracle JDeveloper


26.7.3.1 BPEL Callbacks

If intermediary events need to be propagated to the BPEL process instance, select the **Allow task and routing customization in BPEL callbacks** check box in both the **Advanced** tab of the Human Task window and 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 below:

- `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 request for information, submit information, escalation, reassign, and so on.

- `onSubTaskUpdated` — This callback is invoked for any update to a subtask.

Figure 26–34 shows how a workflow interaction with callbacks is modeled in Oracle BPEL Process Manager. Once 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 26–34 Workflow Interaction Modeling (with Callbacks)

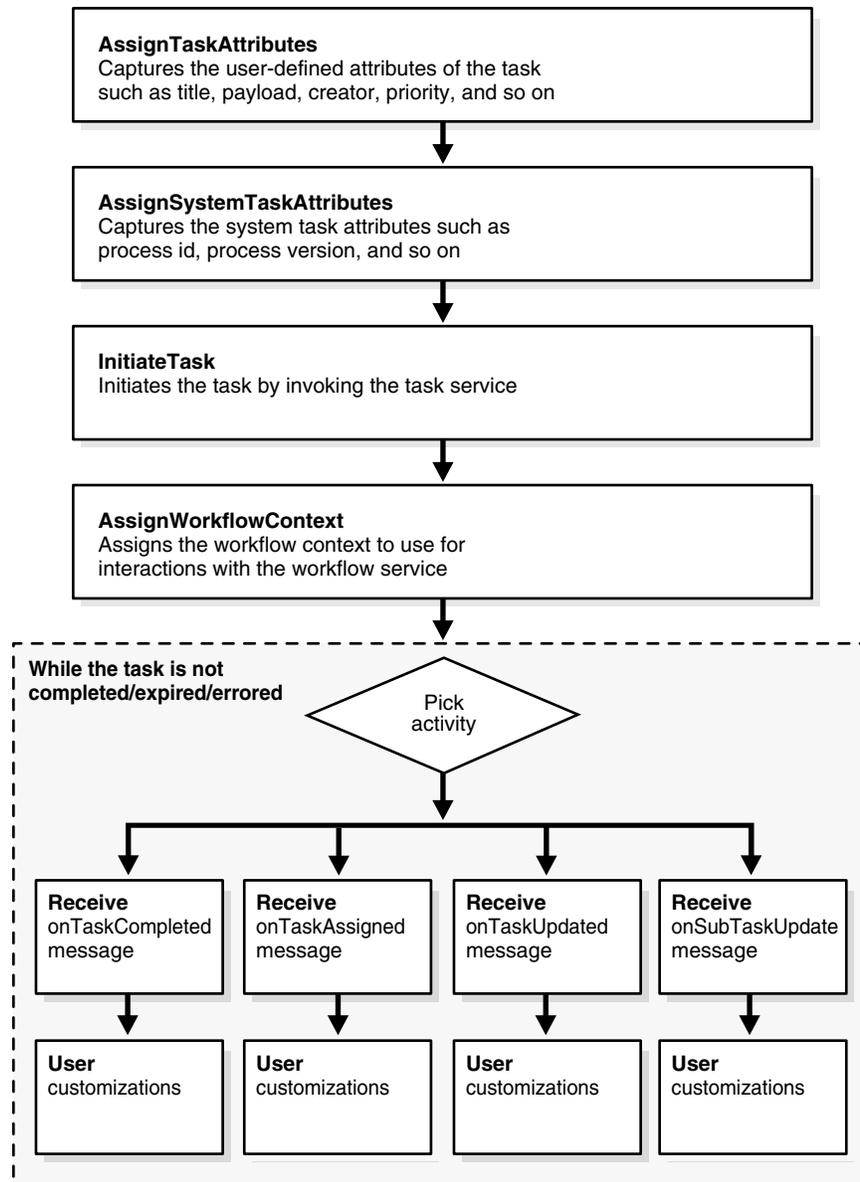
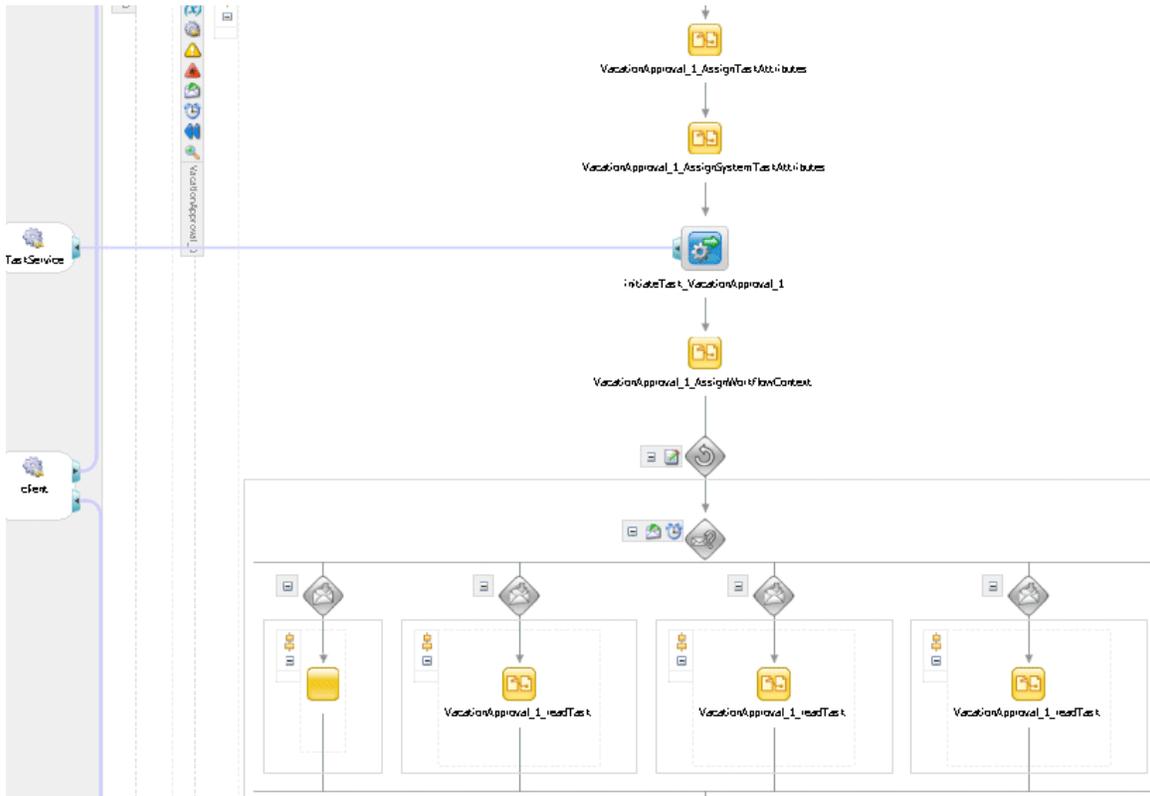


Figure 26–35 shows a workflow interaction in Oracle JDeveloper.

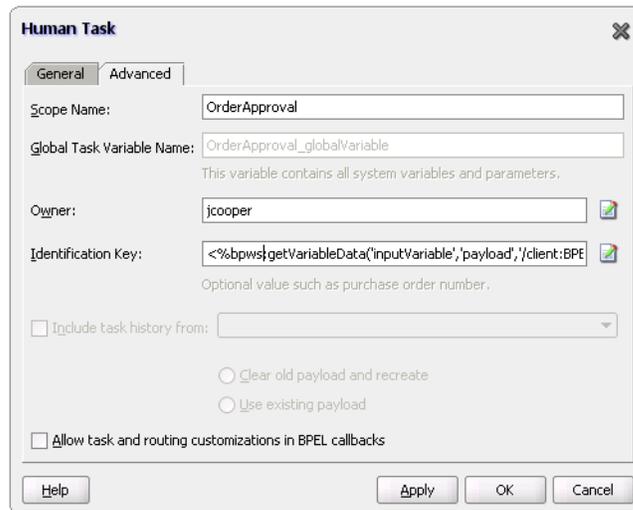
Figure 26–35 Workflow Interaction Modeling (with Callbacks) in Oracle JDeveloper



26.7.4 Defining the Human Task Activity Advanced Features

Figure 26–36 shows the **Advanced** tab.

Figure 26–36 Human Task — Advanced Tab



The **Advanced** tab of the Human Task activity enables you to perform the tasks shown in Table 26–14:

Table 26–14 Human Task - Advanced Tab

For this Field...	See...
Scope Name Global Task Variable Name	Section 26.7.4.1, "Specifying a Scope Name and a Global Task Variable Name" on page 26-78
Owner	Section 26.7.4.2, "Specifying a Task Owner" on page 26-78
Identification Key	Section 26.7.4.3, "Specifying an Identification Key" on page 26-78
Include task history from	Section 26.7.4.4, "Including the Task History of Other Human Tasks" on page 26-78
Allow task and routing customization in BPEL callbacks	Section 26.7.4.5, "Using Task and Routing Customizations in BPEL Callbacks" on page 26-79

26.7.4.1 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.

1. Enter the name for the BPEL scope to be generated in the **Scope Name** field.
This BPEL scope encapsulates the entire interaction with the workflow service and BPEL variable manipulation.
2. Enter the global task variable name in the **Global Task Variable Name** field.
This is the name of the BPEL task variable used for the workflow interaction.

26.7.4.2 Specifying a Task Owner

1. Enter the task owner name in the **Owner** field or click the icon to the right to use the Expression Builder to dynamically specify the owner of this task.
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 window, it defaults to the owner specified here. If an owner is not specified, it defaults to the `bpeladmin` administrator.

26.7.4.3 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 the Oracle BPM Worklist using the identification key. This attribute has no default value.

1. Enter an optional identification key value in the **Identification Key** field.

26.7.4.4 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 where 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.

1. Select the **Include task history from** check box to extend a previous workflow task in the BPEL process. Selecting this check box 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 needs to see the interviewers and the comments they made about the candidate. This process can be modeled using a group vote for the hiring. If the candidate is hired, a database adapter is used to create the entry in the HR database. After this, a simple workflow can include the task history from the group vote 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.

26.7.4.5 Using Task and Routing Customizations in BPEL Callbacks

1. Select the **Allow task and routing customizations in BPEL callbacks** check box to notify the BPEL process using OnMessage callbacks every time a task is routed to a different user or when the task status changes. You must also select the check box of the same name in the **Advanced Settings** section of the Human Task editor shown in [Figure 26–29](#) on page 26-57 in order to update the metadata for callbacks.

In these callbacks, you can call the Task Service to change the routing or update the current assignees. This option impacts the BPEL code generated to interact with the Task Service.

If this option is not selected, the client process gets notified only when the task completes or when it expires or errors out.

2. Click **OK** to close the Human Task window.

3. Go to the Human Task editor for this human task (the `.task` file).
4. Expand the **Advanced Settings** section at the bottom of the editor.
5. Click **Allow task and routing customization in BPEL callbacks**.

This check box *must* be selected to use callbacks. This enables you to update the metadata.

See Also:

- [Section 26.6.8.7, "Allowing Task and Routing Customization in BPEL Callbacks"](#) on page 26-62
- [Section 26.7.3.1, "BPEL Callbacks"](#) on page 26-75

26.7.5 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.

26.7.5.1 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 need to be modified, then this assign can be removed.

26.7.5.2 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.

```
<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>

```

26.8 End-to-End Workflow Examples

Table 26–15 shows the end-to-end workflow examples included in the `soa-samples.zip` file.

In addition to the demonstration features listed in Table 26–15, all samples show the use of worklist applications and workflow notifications.

Table 26–15 *End-to-End Examples*

Sample	Description	Location in soa-samples.zip
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 ADF task forms for the vacation request to act on the task.	<code>workflow\workflow-100-VacationRequest</code>
Help Desk Request	Provides a simple workflow sample using ADF task forms for task approval.	<code>workflow\workflow-101-HelpDeskRequest</code>
Sales Quote Request	Provides a complex workflow sample with chaining of multiple tasks.	<code>workflow\workflow-102-SalesQuote</code>

Table 26–15 (Cont.) End-to-End Examples

Sample	Description	Location in soa-samples.zip
Expense Application	Provides a sample that integrates workflow with ADF business components (BC). Events are raised to the BPEL process and the human workflow is invoked for task approval.	workflow\workflow-103-ExpenseApp
Contract Approval	Provides a sample of approving a contract. This sample uses digital signatures for tasks.	workflow\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\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\workflow-106-IterativeDesign
Office Integration	Provides a sample in which Microsoft Excel attachments are enabled with workflow notifications.	workflow\workflow-107-OfficeIntegration

26.8.1 Help Desk Request Example

This describes how to create a help desk request business process. In this example, a customer files a help desk ticket, which is assigned to a help desk agent who either resolves this request or routes it to other agents for resolution.

This example illustrates the usage of BPEL process and human task service components in a SOA composite application. This example also describes how to create ADF task forms that the agent uses to act on the task.

This example highlights the use of the following:

- Using the SOA Composite Editor
- Modeling a single approval workflow using Oracle JDeveloper
- Creating an ADF-based Oracle BPM Worklist
- Using the Oracle BPM Worklist to view and respond to the task

26.8.2 Prerequisites

This example assumes 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 JDeveloper—the environment for designing and deploying BPEL processes—is also assumed.

26.8.3 Modeling the Help Desk Request

In this tutorial, you create a new application and SOA project and design the human task service component, a single approver workflow in which the order is either resolved or unresolved by a single help desk employee. You also create a second application and project in which you design an ADF-based Oracle BPM Worklist from which to act upon the help desk request.

This section contains these tasks:

- [Section 26.8.3.1, "Creating an Application"](#)
- [Section 26.8.3.2, "Creating the SOA Project"](#)
- [Section 26.8.3.3, "Create the Human Task Service Component"](#)
- [Section 26.8.3.4, "Designing the Human Task"](#)
- [Section 26.8.3.5, "Associating the Human Task and BPEL Process Service Components"](#)
- [Section 26.8.3.6, "Deploying the SOA Composite Application"](#)
- [Section 26.8.3.7, "Initiating the Process Instance"](#)
- [Section 26.8.3.8, "Creating an Oracle BPM Worklist SOA Project"](#)
- [Section 26.8.3.9, "Designing the Task Display Form"](#)
- [Section 26.8.3.10, "Resolving the Task in Oracle BPM Worklist"](#)

26.8.3.1 Creating an Application

You first create an application for the SOA project.

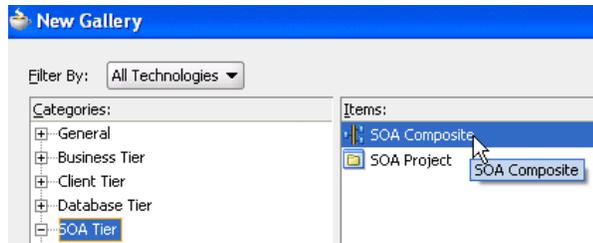
1. Start Oracle JDeveloper.
2. Select **New > Application** from the **File** main menu.
3. Enter the following values:

Field	Value
Application Name	Enter an application name (for this example, HelpDeskRequestApp).
Application Template	Select No Template [All Technologies] .

4. Accept the default values for all remaining settings, and click **OK**.
5. The Create Project window appears.
6. Enter a name for the project (for this example, **HelpDeskRequestComposite**), and click **OK**.

26.8.3.2 Creating the SOA Project

1. Locate the `HelpDeskRequestSCAApp.zip` file in the `workflow\workflow-101-HelpDeskRequest` directory of the `soa-samples.zip` file.
2. Unzip the `HelpDeskRequestSCAApp.zip` file into the `workflow\workflow-101-HelpDeskRequest` directory.
3. Right-click the project name in the **Application Navigator** and select **New**.
4. Select **SOA Tier** in the **Categories** list.
5. Double-click **SOA Composite** in the **Items** field.



The Create SOA Composite window appears.

6. Select **Composite with BPEL**. This creates an SOA project with a BPEL process.
7. Click **OK**.

The Create BPEL Process window appears.

8. Enter the following values:

Field	Value
Project Name	Enter HelpDeskRequestProcess .
Template	Select Asynchronous BPEL Process .
Expose as Composite Service	Select the check box. After process creation, note the SOAP adapter service that appears in the Services swim lane. This service provides the entry point to the SOA composite application from the outside world.

9. Click the **Browse** icon to the right of the **Input** field.

The Type Chooser window appears.

10. Click the **Import Schema File** icon.

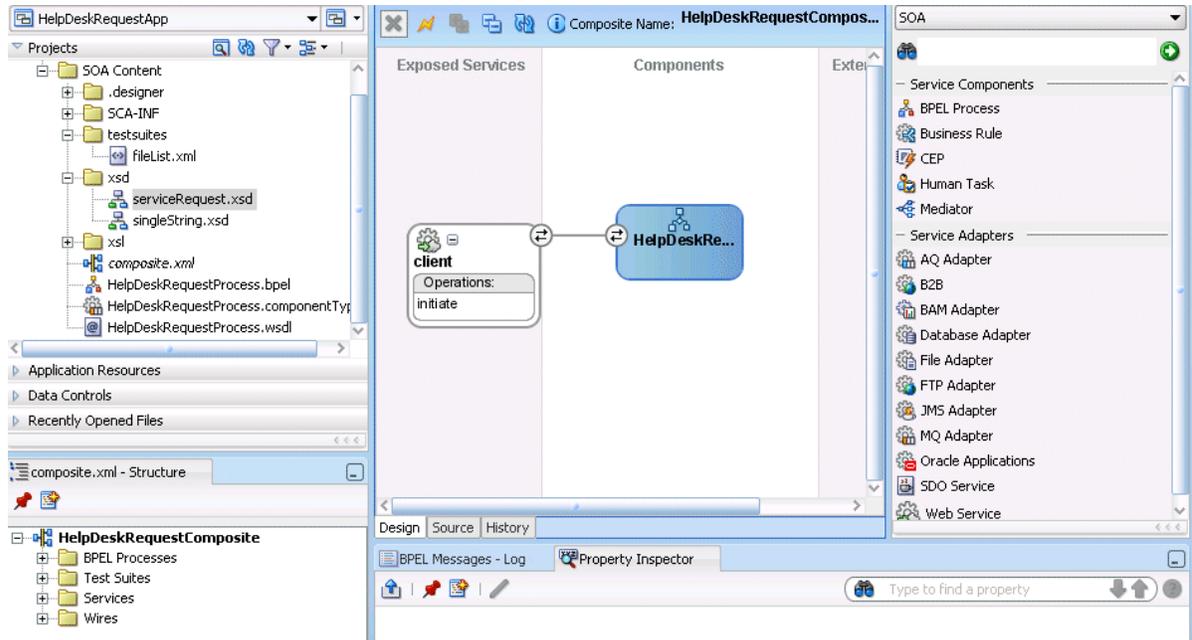
The Import Schema File window appears.

11. Click the **Browse** icon.
12. Select the `serviceRequest.xsd` file located in the `workflow\workflow-101-HelpDeskRequest` directory.
13. Click **OK**.

You are returned to the Type Chooser window.

14. Expand and select **Project Schema Files > serviceRequest.xsd > HelpDeskRequest**.
15. Click **OK**.
16. Click the **Browse** icon to the right of the **Output** field.
17. Repeat Steps 14 through 15.
18. Click **OK**.

The BPEL process displays in the SOA Composite Editor.



26.8.3.3 Create the Human Task Service Component

1. Drag and drop a **Human Task** into the canvas workspace from the **SOA** list of the **Component Palette**.

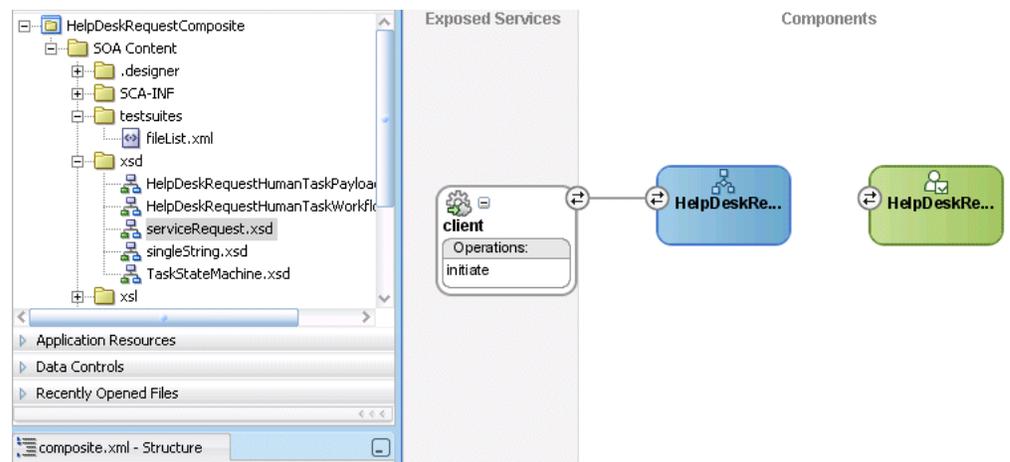
The Create Human Task window appears.

2. Enter the following details.

Field	Value
Human Task Definition Name	Enter <code>HelpDeskRequestHumanTask</code> .
Expose as Composite Service	Do <i>not</i> select the check box.

3. Accept the default settings for all other fields.
4. Click **OK**.

The **Human Task** icon appears in the SOA Composite Editor canvas workspace.



- Double-click the **Human Task** icon.

The Human Task editor appears.

26.8.3.4 Designing the Human Task

- Enter the following values:

Field	Value
Title	Enter Help desk request for . Note: Ensure that you enter a blank space after for .
Outcomes	Enter RESOLVED,UNRESOLVED
Description	Leave unspecified.
Priority	Accept the default value.
Category	Leave unspecified.
Owner	Leave unspecified.

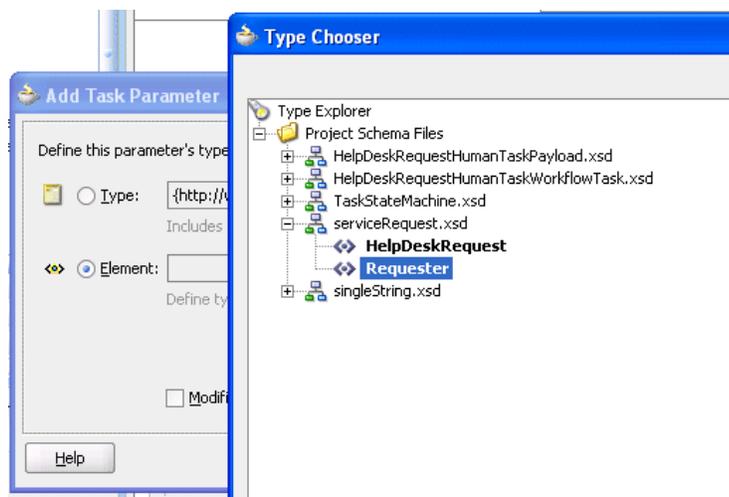
- Click the **Add** icon on the right side of the **Parameters** section to specify the task payload.

The Add Task Parameter window is displayed.

- Click **Element** and then the **Browse** icon to the right of this field.

The Type Chooser window appears.

- Expand and select **Project Schema Files** > **serviceRequest.xsd** > **Requester**, and click **OK**.

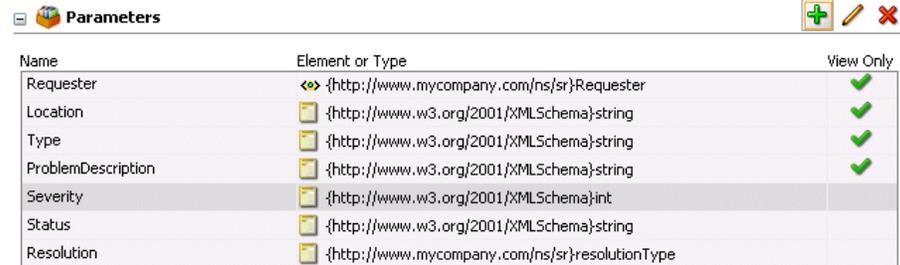


- Ensure that the **Modifiable via worklist** check box is *not* selected.
- Click **OK** on the Add Task Parameter window.
- Click the **Add** icon again on the right side of the Parameters section.
- Click **Type** and then the **Browse** icon to the right of this field.
- Expand and select **XML Schema Simple Types** > **string**.
- Click **OK**.

11. Enter **Location** in the **Name** field
12. Ensure that the **Modifiable via worklist** check box is *not* selected, and click **OK**.
13. Repeat Steps 7 through 11 and create the remaining parameters with the following values:

Name	Select This Type...	Select the Modifiable via Worklist Check Box?
Type	XML Schema Simple Types > string	No
ProblemDescription	XML Schema Simple Types > string	No
Severity	XML Schema Simple Types > int	Yes
Status	XML Schema Simple Types > string	Yes
Resolution	Project Schema Files > serviceRequest.xsd > resolutionType	Yes

When complete, the **Parameters** section looks as follows:



Name	Element or Type	View Only
Requester	{http://www.mycompany.com/ns/sr}Requester	✓
Location	{http://www.w3.org/2001/XMLSchema}string	✓
Type	{http://www.w3.org/2001/XMLSchema}string	✓
ProblemDescription	{http://www.w3.org/2001/XMLSchema}string	✓
Severity	{http://www.w3.org/2001/XMLSchema}int	✓
Status	{http://www.w3.org/2001/XMLSchema}string	✓
Resolution	{http://www.mycompany.com/ns/sr}resolutionType	✓

14. Click the **Add** icon on the right side of the **Assignment and Routing Policy** section.

The Add Participant Type window appears.

15. Select **Single Approver** from the **Type** list.
This participant type acts alone on the task.
16. Enter **Help Desk Agent** in the **Label** field.
17. Enter **jstein** in the **User Id(s)** field.
18. Expand **Advanced** at the bottom of the window.
19. Select the **Allow this participant to invite other participants** check box to enable adhoc routing.
20. Click **OK**.
21. Click the + sign to expand the **Expiration and Escalation Policy** section.
22. Enter the following details:

Field	Value
Drop-Down List	Select Escalate after.
Fixed Duration Day	Select this button.
Day	Select 1.
Minutes	Select 10.

Field	Value
Maximum Escalation Levels	Enter 3.
Highest Approver Title	Select CEO.

23. Expand the **Notification Settings** section.

Notifications are enabled by default for assignees, initiator, and owner.

24. Select the **Make email messages actionable** check box.

This enables e-mails to be sent to assignees with an actionable link.

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

26. Click the **x** next to **HelpDeskRequestHumanTask.task** above the editor to close the Human Task editor.

26.8.3.5 Associating the Human Task and BPEL Process Service Components

1. Double-click the BPEL process service component in the SOA Composite Editor.

The BPEL process displays in Oracle JDeveloper.

2. Select **BPEL** from the **Component Palette**.

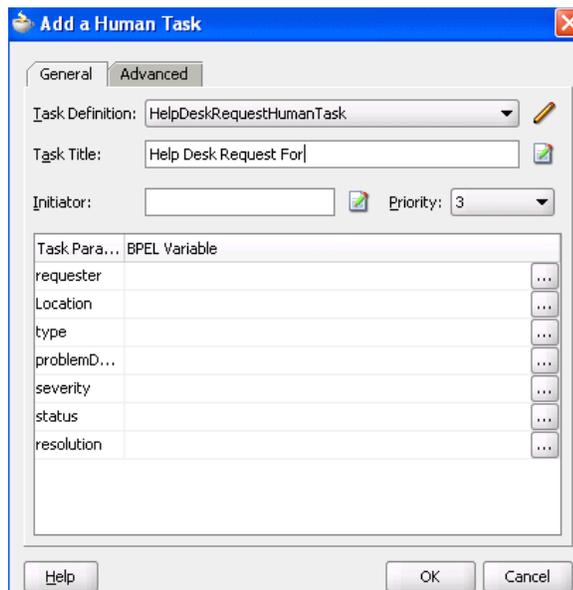
3. Expand **BPEL Activities and Components**.

4. Drag and drop a **Human Task** below the **receiveInput** receive activity.

The Add a Human Task window appears.

5. If it is not currently displaying, select **HelpDeskRequestHumanTask** from the **Task Definition** list.

The window refreshes to display additional fields.



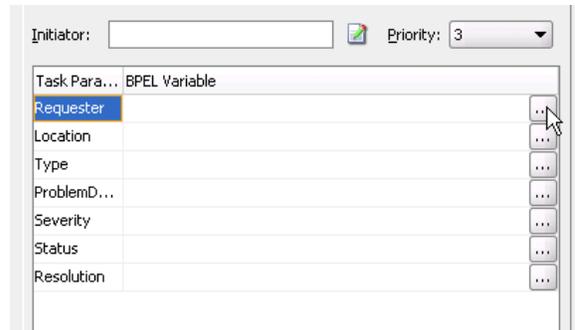
6. Note that **Help desk request for** appears in the **Task Title** field. You entered this name in Step 1 on page 26-86.

7. Click the **XPath Expression Builder** icon to the right of this field to display the Expression Builder window.

8. Press **Ctrl** and then the **space bar** in the **Expression** field and select the following XPath expression:

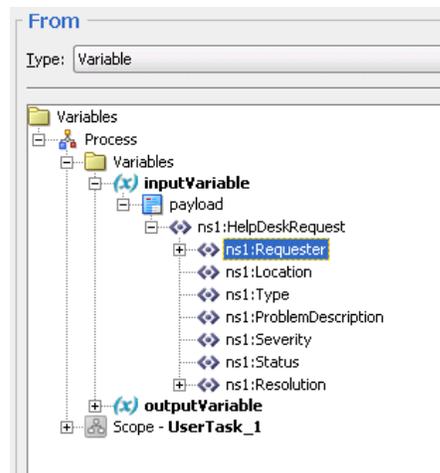

```
bpws:getVariableData('inputVariable','payload','/ns1:HelpDeskRequest/ns1:requester/ns1:ID')
```
9. Click **Insert Into Expression**.
10. Click **OK**.

The XPath expression displays in the **Task Title** field.
11. Click the **BPEL Variable** entry for the **requester** parameter.



The Task Parameters window appears.

12. Select **Variable** from the **Type** list.
13. Expand **Process > Variables > inputVariable > payload > ns1:HelpDeskRequest > ns1:Requester**.

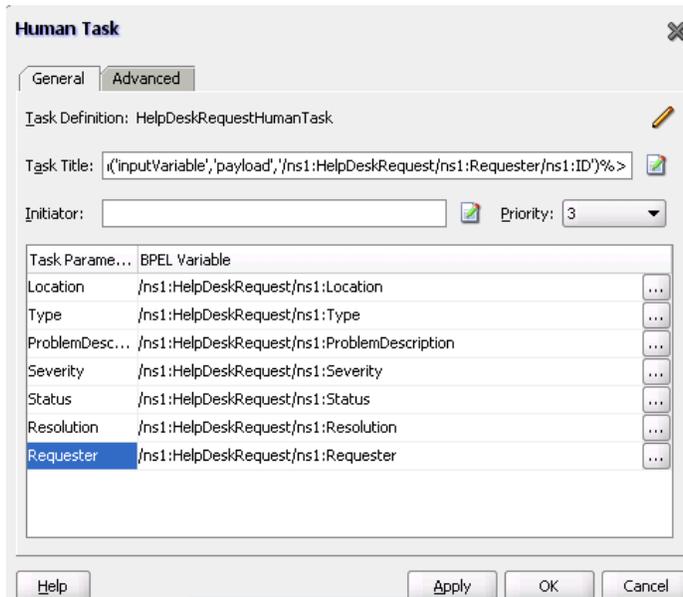


14. Click **OK**.
15. Repeat Step 11 through 14 to select the following BPEL variables.

Variable	Select Process > Variables > inputVariable > payload > ns1:HelpDeskRequest >...
Location	ns1:location
Type	ns1:type
ProblemDescription	ns1:problemDescription

Variable	Select Process > Variables > inputVariable > payload > ns1:HelpDeskRequest >...
Severity	ns1:severity
Status	ns1:status
Resolution	ns1:resolution

When complete, the window looks as follows:



16. Click **OK** to close the Add a Human Task window.

The `HelpDeskRequestTaskService_1` partner link now appears.

26.8.3.6 Deploying the SOA Composite Application

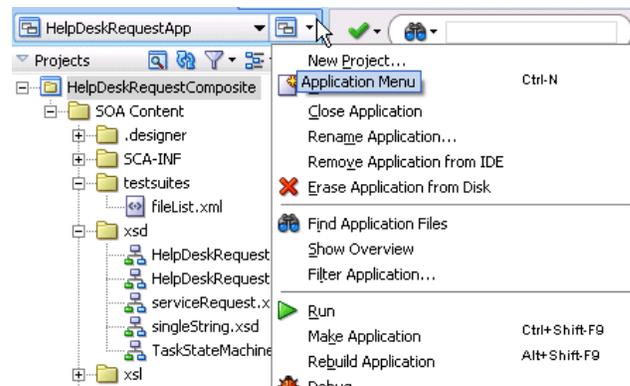
See [Section 3.4, "Deploying Applications"](#) on page 3-2 for instructions on how to deploy the SOA composite application.

26.8.3.7 Initiating the Process Instance

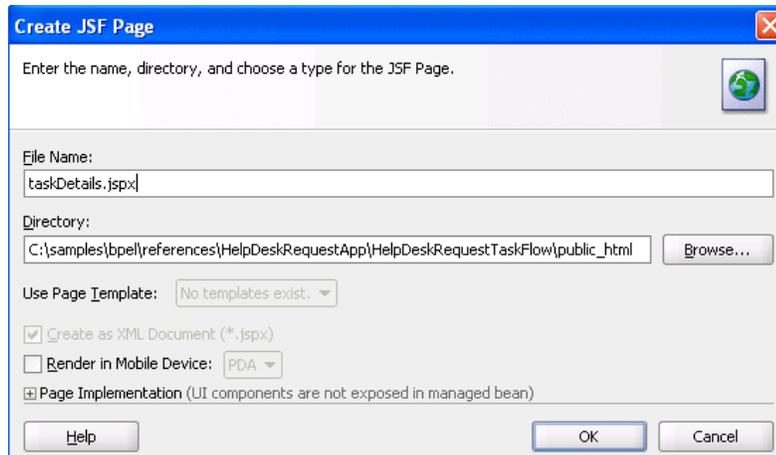
1. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for instructions on accessing the page for initiating the process instance.
2. Copy and paste the contents of `workflow\workflow-101-HelpDeskRequest\serviceRequest.xml` as input.

26.8.3.8 Creating an Oracle BPM Worklist SOA Project

1. Select the **Application Menu** for the `HelpDeskRequestApp` application you created in [Section 26.8.3.1, "Creating an Application"](#) on page 26-83.



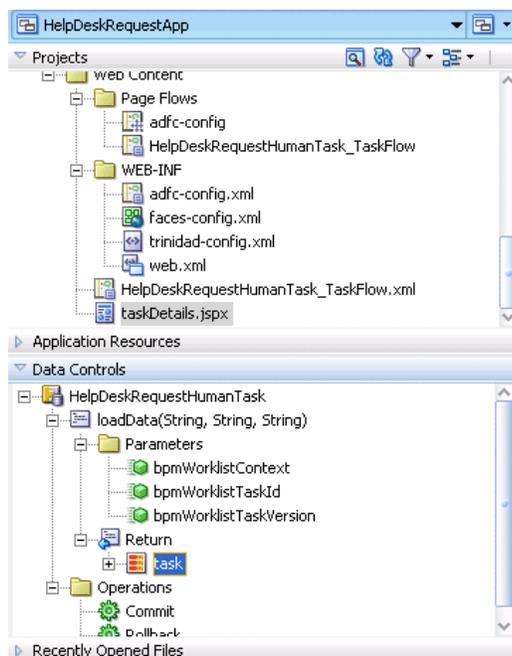
2. Select **New Project**.
3. Select **Empty Project** and click **OK**.
4. Enter **HelpDeskRequestTaskFlow** in the **Project Name** field.
5. Accept the default setting for all other fields.
6. Click **OK**.
7. Click **Cancel** on the Create Project window.
8. Select **Save All** from the **File** main menu.
9. Select the **Application Menu** list for the application you just created.
10. Select **New Project** from the list that appears.
The New Gallery window appears.
11. Select **Empty Project** from the **Items** list.
12. Click **OK**.
13. Enter **HelpDeskRequestTaskFlow** in the **Project Name** field, and click **OK**.
14. Right-click **HelpDeskRequestTaskFlow** in the **Application Navigator** and select **New**.
15. Select **Web Tier > JSF > ADF Task Flow Based on Human Task**.
16. Click **OK**.
The SOA Resource Lookup window appears.
17. Select the **HelpDeskRequestHumanTask.task** file you created in previous sections.
This creates the ADF data controls and displays the Create ADF Task Flow window.
18. Click **OK**.
19. Select **Save All** from the **File** main menu.
20. Double-click the **taskDetails** icon that appears.
This starts the Create JSF JSP window to create the JSPX file.



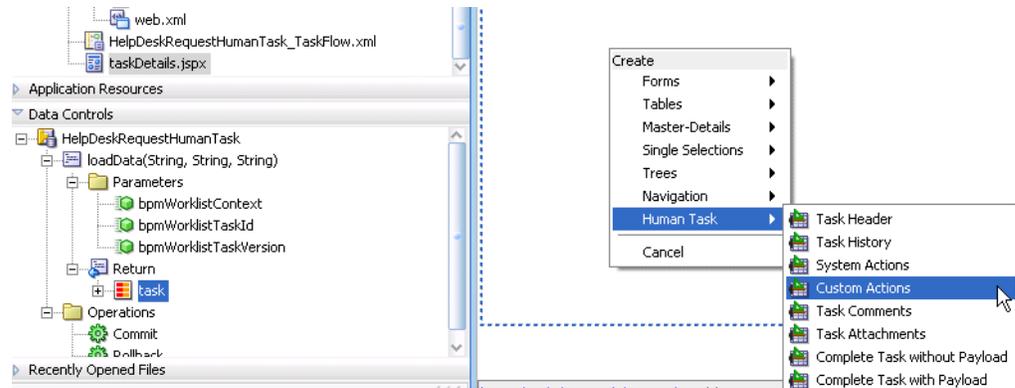
21. Click **OK**.
22. Select **Save All** from the **File** main menu.

26.8.3.9 Designing the Task Display Form

1. Go to the **Data Controls** panel in the **Application Navigator**.
2. Expand **loadData(String, String, String) > Return > task**.



3. Drag and drop a **task** icon into the JSPX window.
A menu appears.
4. Select **Human Task > Custom Actions**.



5. Click **OK** when prompted in the Edit Action Bindings window.

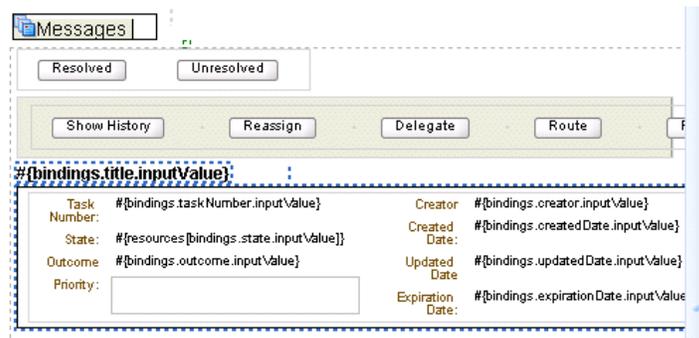
This creates two buttons: **Resolved** and **Unresolved**.

6. Drag and drop a second **task** icon into the JSPX window.
7. Select **Human Task > System Actions**.

This creates two additional buttons: **Delegate** and **Route**.

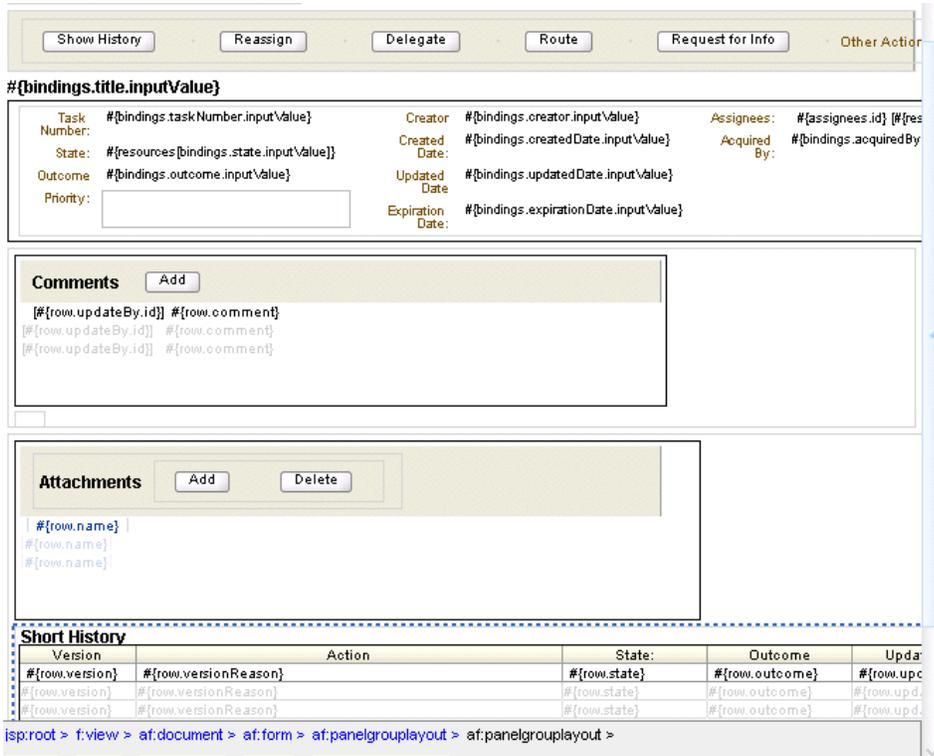
8. Drag and drop a third **task** icon into the JSPX window.
9. Select **Human Task > Task Header**.

With each element that you drag and drop, the JSPX window continues to expand:

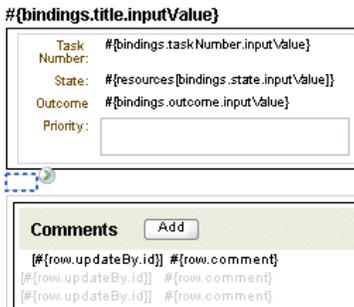


10. Drag and drop additional **task** icons into the JSPX window and select these options:
 - **Human Task > Task Comments.**
 - **Human Task > Task Attachments.**
 - **Human Task > Task History**

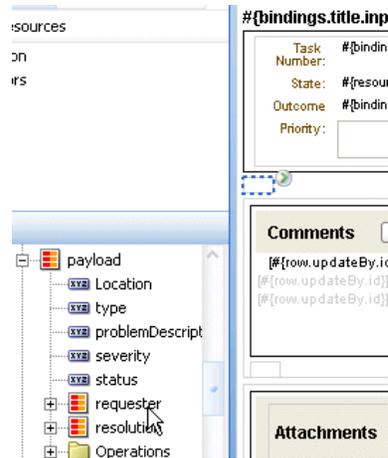
When complete, the bottom part of the JSPX window appears as follows:



11. Select **ADF Faces** from the **Component Palette**. If this entry does not appear in the Component Palette, select **All Pages** from the list.
12. Drag and drop **Panel Group Layout** between the **Header** and **Comment** sections. This displays a small blue box between the two sections.

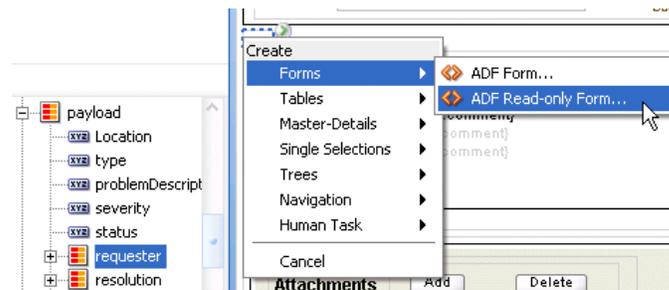


13. Return to the **Data Controls** panel in the **Application Navigator**.
14. Expand **payload**.
15. Drag and drop **requester** in front of the small blue box that represents the **Panel Group Layout**.



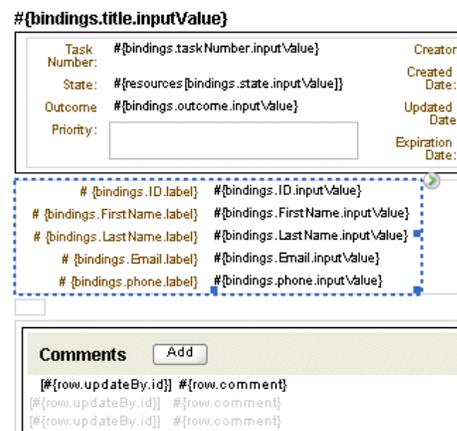
A menu appears.

16. Select the read-only drop handler for **requester** by selecting **Forms > ADF Read-only Form**.

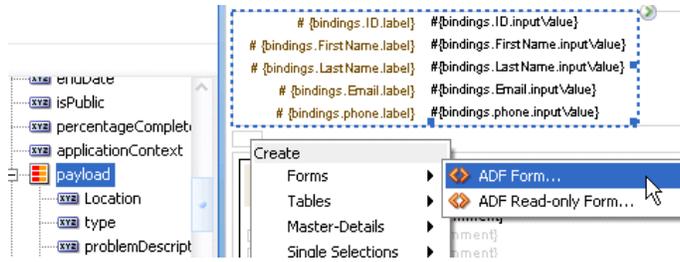


17. Click **OK** on the **Edit Form Fields** window.

This creates the following section between the **Header** and **Comments** sections:



18. Drag and drop **payload** below the **Panel Group Layout** and select **ADF Form > Forms**.



19. Click OK on the **Edit Form Fields** window.

The window appears as follows:

{bindings.title.inputValue}

Task Number: <input type="text" value="# {bindings.taskNumber.inputValue}"/>	Creator: <input type="text" value="# {bindings.creator.inputValue}"/>	Assignees:
State: <input type="text" value="# {resources[bindings.state.inputValue]}"/>	Created Date: <input type="text" value="# {bindings.createdDate.inputValue}"/>	Acquired By:
Outcome: <input type="text" value="# {bindings.outcome.inputValue}"/>	Updated Date: <input type="text" value="# {bindings.updatedDate.inputValue}"/>	
Priority: <input type="text"/>	Expiration Date: <input type="text" value="# {bindings.expirationDate.inputValue}"/>	

{bindings.ID.label}

{bindings.FirstName.label}

{bindings.LastName.label}

{bindings.Email.label}

{bindings.phone.label}

{bindings.Location.label}

{bindings.type.label}

{bindings.problemDescription.label}

{bindings.severity.label}

{bindings.status.label}

Comments

{row.updateBy.id}

{row.updateBy.id}

20. Drag and drop **resolution** below the payload and select **Forms > ADF Form**.

21. Click **OK** when prompted on the **Edit Form Fields** window.

The JSPX window appears as follows:

# {bindings.ID.label}	# {bindings.ID.inputValue}
# {bindings.FirstName.label}	# {bindings.FirstName.inputValue}
# {bindings.LastName.label}	# {bindings.LastName.inputValue}
# {bindings.Email.label}	# {bindings.Email.inputValue}
# {bindings.phone.label}	# {bindings.phone.inputValue}

# {bindings.Location.label}	<input type="text"/>
# {bindings.type.label}	<input type="text"/>
# {bindings.problemDescription.label}	<input type="text"/>
# {bindings.severity.label}	<input type="text"/>
# {bindings.status.label}	<input type="text"/>

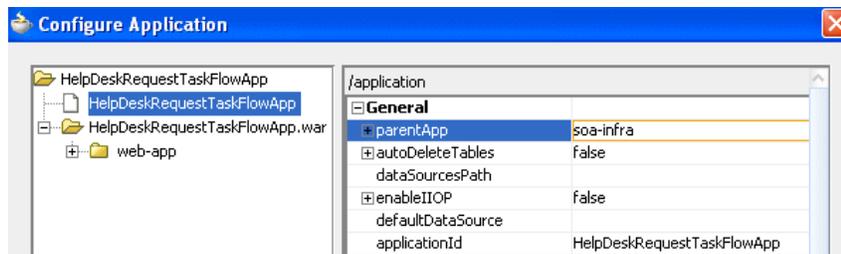
# {bindings.comment1.label}	<input type="text"/>
# {bindings.resolvedBy.label}	<input type="text"/>

Comments	<input type="button" value="Add"/>
<pre> [#{row.updateBy.id}] #{row.comment} [#{row.updateBy.id}] #{row.comment} [#{row.updateBy.id}] #{row.comment} </pre>	

22. Go to folder hierarchy in the **Application Navigator**.
23. Select and double-click **Application Sources > WEB-INF > taskflow.properties**.
24. Add the following line at the bottom of this file:

```
human.task.lookup.type=LOCAL
```
25. Right-click the **HelpDeskRequestTaskFlow** project name and select **Project Properties**.
26. Select **Deployment** and click **New**.

The Create Deployment Profile window appears.
27. Select WAR File from the **Archive Type** list.
28. Enter **HelpDeskRequestTaskFlowApp** in the **Name** field.
29. Click **OK**.
30. Click **Edit** and select the **Specify Java EE Web Context Root** radio button.
31. Enter **HelpDeskRequestTaskFlowApp** in the field immediately below.
32. Click **OK** to close the WAR Deployment Profile Properties window and the Project Properties window.
33. Right-click **HelpDeskRequestTaskFlowApp** and select **Deploy > HelpDeskRequestTaskFlowApp > to > application_server_connection**. If you do not have a connection, select **New Connection** and provide responses to the prompts in the Application Server Connection wizard.
34. The **Configure Application** window appears.
35. Select **HelpDeskRequestTaskFlowApp > parentApp as soa-infra**.



36. Click OK.

26.8.3.10 Resolving the Task in Oracle BPM Worklist

1. Go to the Oracle BPM Worklist:

<http://localhost:8888/integration/worklistapp>

2. Log in as `jstein/welcome1`.

3. Select **HelpDeskRequestSCAApp**.

Designing Task Display Forms for Human Tasks

As described in [Chapter 26, "Designing 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.

The task display form renders custom actions and system actions. Custom actions are created using task outcomes. The task display form is created using ADF task flows in Oracle JDeveloper. This chapter describes how to design and customize task display forms.

This chapter contains the following sections:

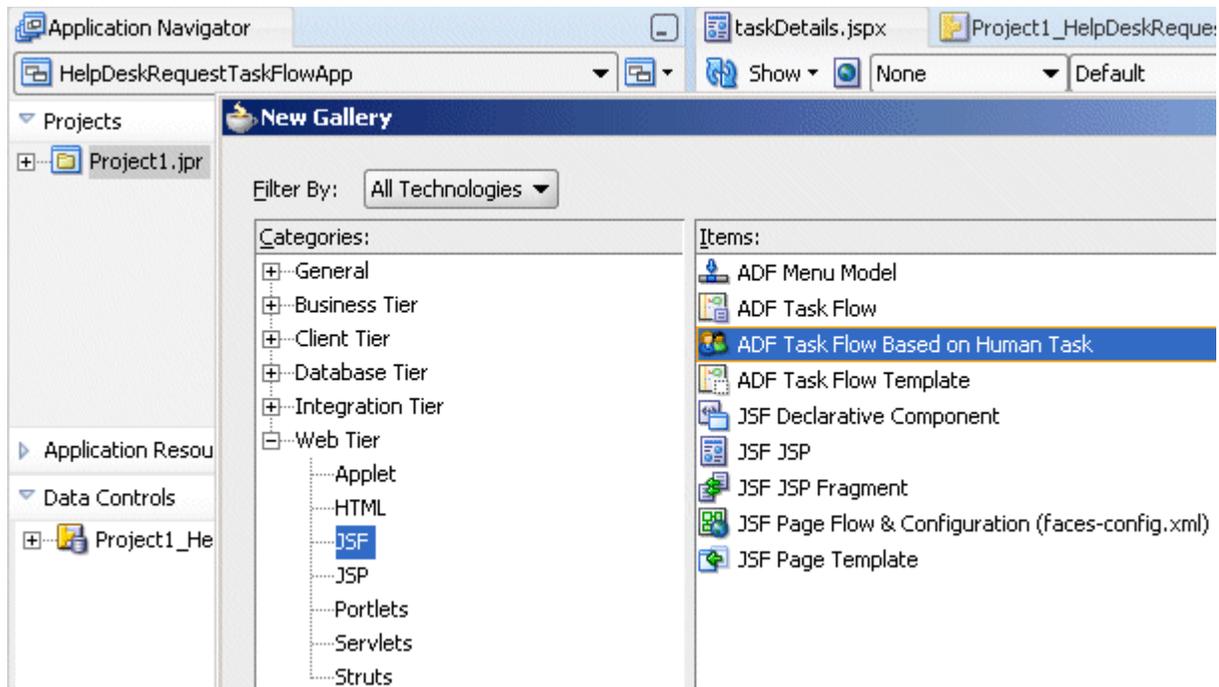
- [Section 31.1, "Introduction to the Task Display Form"](#)
- [Section 31.2, "Associating the Task Flow with the Task Service"](#)
- [Section 31.3, "Creating an ADF Task Flow Based on a Human Task"](#)
- [Section 31.4, "Creating a Task Display Form"](#)
- [Section 31.5, "Creating an E-Mail Notification"](#)
- [Section 31.6, "Deploying a Composite Application with a Task Flow"](#)
- [Section 31.7, "Displaying a Task Display Form in the Worklist"](#)
- [Section 31.8, "Troubleshooting the Task Display Form"](#)

31.1 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 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 same Oracle JDeveloper visual editor where you created the SOA composite containing the human task. [Figure 31-1](#) shows the Oracle JDeveloper **ADF Task Flow Based on Human Task** option where you start creating a task display form.

Figure 31–1 ADF Task Flow Based on a Human Task, in Oracle JDeveloper



31.2 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 to connect to the workflow service engine at execution time to retrieve task content and act on tasks.

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 the resource discovery service in MAS. However, you can override these by explicitly specifying the URL and port information here.

31.3 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 following ways:

- Within the same application that contains the human workflow task
- In a separate application

31.3.1 How to Create an ADF Task Flow Within the Same Application as the Human Task

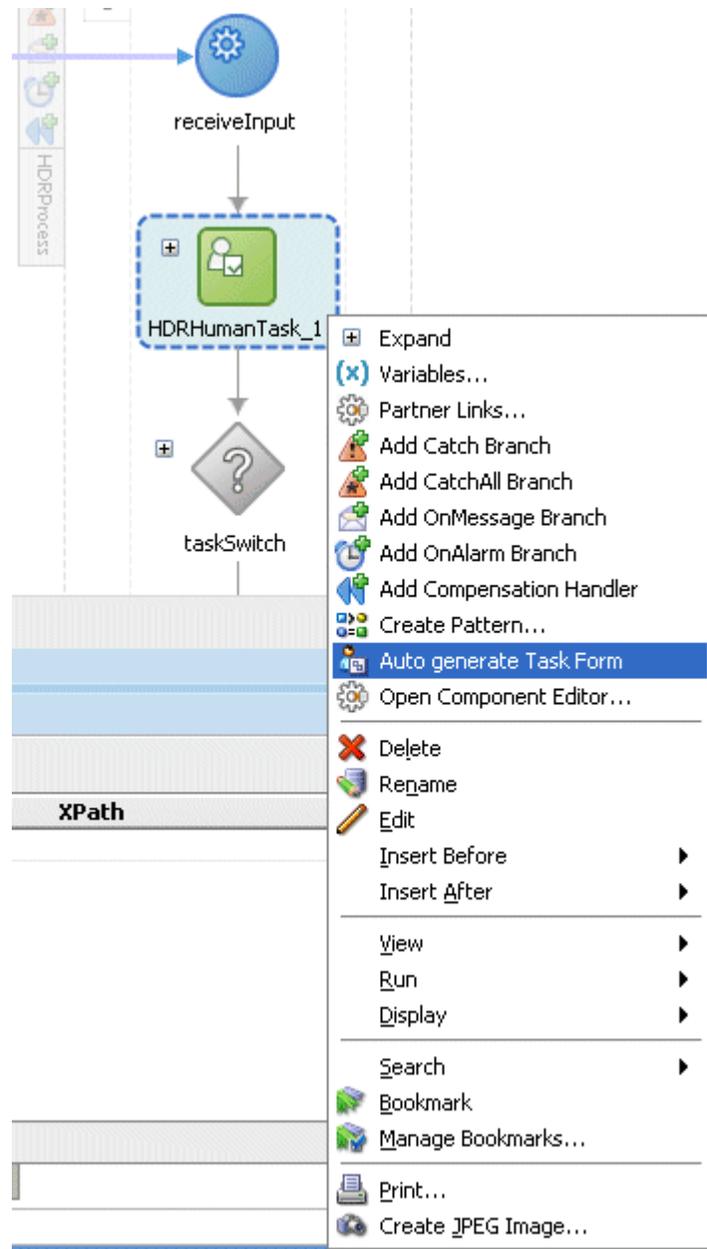
[Summary to come]

To create an ADF task flow within the same application as the human task

1. Open the BPEL process (in this example, `HelpDeskRequestProcess.bpel`) within the SOA composite application.

- Right-click the human task activity (in this example, **HelpDeskRequestHumanTask_1**) and select **Auto generate Task Form**, as shown in [Figure 31-2](#).

Figure 31-2



- Provide a project name.

The **taskDetails.jspx** window appears.

To continue creating the task display form, see Step 1 in [Section 31.4.3, "How to Create a Task Display Form Using Individual Drop Handlers."](#)

31.3.2 How to Create an ADF Task Flow Within the Same Composite Application as the Human Task

The ADF Task Flow Based on Human Task function prompts you for the `.task` file to use for the ADF task flow, which you find using the file browser or the resource catalog. 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 26, "Designing Human Tasks,"](#) for how to create the `.task` file for the help desk request example used in the following steps. When the `.task` file is selected, data controls are automatically created based on the task parameters and outcomes.

You can also follow these steps to create the task flow in the *same* application. Instead of steps 1 and 2, select the application that contains the human task (for this example, `HelpDeskRequestApp`).

To create an ADF task flow in a separate application:

1. In the Application Navigator, select **New Application**.
2. Provide an application name (for this example, `HelpDeskRequestTaskFlowApp`).
3. Create an empty project (for this example, `HelpDeskRequest`).
4. Right-click the project and select **New**.
5. From **Filter By**, select **All Technologies**.
6. Expand **Web Tier** and select **JSF**.
7. Select **ADF Task Flow Based on Human Task** and click **OK**.
8. In the SCA Resource Lookup dialog, find the `.task` file where you defined the human task (for this example, `HelpDeskRequestHumanTask.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.

This displays the Create ADF Task Flow dialog and creates the data controls.

9. In the Create ADF Task Flow dialog, accept the defaults and click **OK**.

The `taskDetails` icon appears in the visual editor.

10. From **File**, select **Save All**.

To continue creating the task display form, see Step 1 in [Section 31.4.3, "How to Create a Task Display Form Using Individual Drop Handlers."](#)

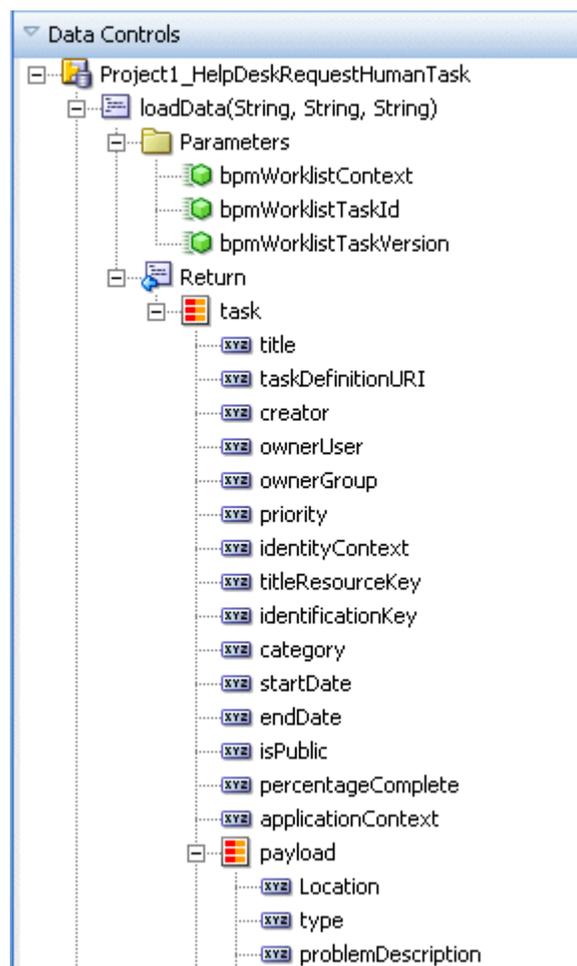
31.3.3 What Happens When You Create an ADF Task Flow Based on a Human Task

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 rapidly create interface regions to display 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 31-3](#). The data controls for the task (represented by the **task** node in the figure) have custom drop handlers to render specific regions of the task, for example, the task header, system actions, custom actions, comments, and so on. In addition, the standard ADF drop handlers, used to create tables, forms, and so on, are available for rendering the payload.

Figure 31-3 The Task Collection in the Data Controls Panel



31.4 Creating a Task Display Form

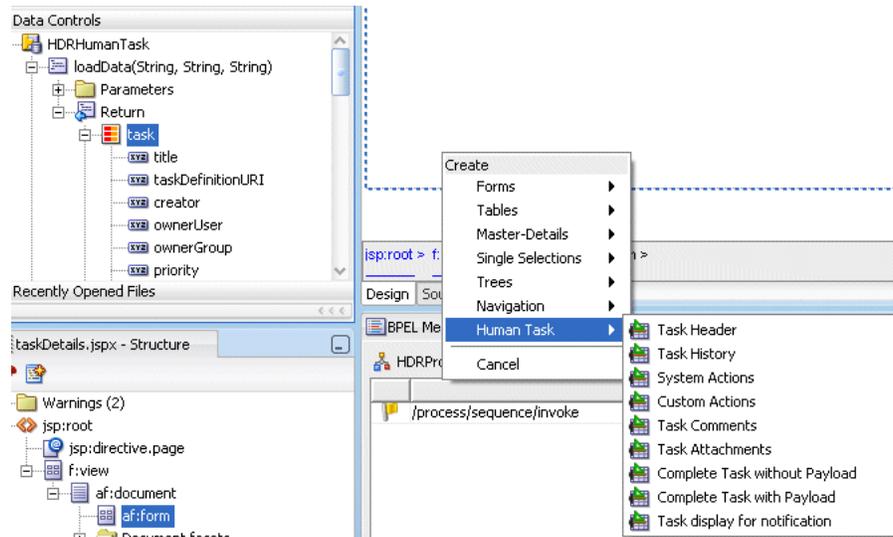
Creating a task display form is an iterative process in that you repeatedly drag the data control for the task (the **task** node in the Data Controls panel) into the visual editor and then use Oracle ADF drop handlers to build the regions of the task display form. The following Oracle ADF drop handlers are available:

- Custom drop handlers
 - These render specific regions of the task details page: the task header, task history, system actions, custom actions, task comments, and task attachments.
- Standard drop handlers
 - These render the payload region of the task details page, including forms and tables.

31.4.1 Custom Drop Handlers

Custom drop handlers appear in the context menu of the visual editor, as shown in Figure 31–4.

Figure 31–4 Custom Drop Handlers for Creating the Task Display Form for a Human Task



The custom drop handlers are designed to autogenerate the following regions of the task details page:

- Task header
- Task history
- System actions
- Custom actions
- Task comments
- Task attachments
- Complete task without payload
- Complete task with payload
- Task display for notification

31.4.1.1 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 31–5 shows an example of header information for the help desk request example.

Figure 31–5 Header Information

#[bindings.title.input\value]			
Task Number:	#[bindings.taskNumber.input\value]	Creator:	#[bindings.creator.input\value]
State:	#[resources[bindings.state.input\value]]	Created Date:	#[bindings.createdDate.input\value]
Outcome:	#[bindings.outcome.input\value]	Updated Date:	#[bindings.updatedDate.input\value]
Priority:	<input type="text"/>	Expiration Date:	#[bindings.expirationDate.input\value]

31.4.1.2 Task History

The short history appears on the task details page and lists the approval history. It includes the following attributes:

- state
- outcome
- comments
- updatedDate
- version
- versionReason
- updatedBy
- Id

31.4.1.3 System Actions

The following system actions are available as buttons on the worklist. The actual buttons 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 approve and reject appear.

- Show History—This action shows the details of the task history.
- 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.
- Reassign—Managers can reassign a task to reportees. A user with BPMWorkflowReassign privileges can reassign a task to anyone.
- Delegate—A task can be delegated to another user or group. The delegated task will show up 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.

- Request More 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.
- Go—This button is used to update a system action that you selected from a list. The list can contain the following simple actions:
 - Escalate—An escalated task is assigned to the user’s manager. The Comments area is available for an optional comment.
 - Pushback—This action sends a task up one level in the workflow to the previous assignee.
 - 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.
 - 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. [Cross-reference to come. Will point to screen and instructions in the SOA Admin Guide.]

A renewal appears in the task history. The Comments area is available for an optional comment.
 - Suspend/Resume—These options 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.
 - 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.
- 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 will appear in the worklist. [Figure 31–6](#) shows an example of system action buttons on a task display form.

Figure 31–6 System Action Buttons



31.4.1.4 Custom Actions

Custom actions create buttons on the task details page that represent actions defined in the human task. [Figure 31–7](#) shows an example of custom action buttons for the help desk request example.

Figure 31–7 Custom Action Buttons

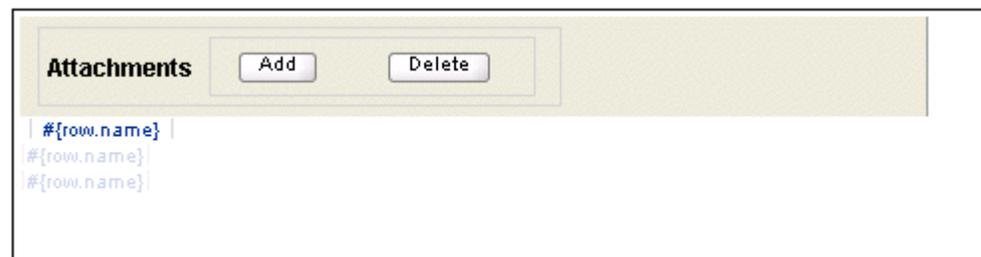
31.4.1.5 Task Comments

A trail of comments and the comment writer's user name is maintained throughout the life cycle of a task. [Figure 31–8](#) shows the comment region for the help desk request example.

Figure 31–8 Comment Information

31.4.1.6 Task Attachments

Files or reference URLs associated with a task can be added by any of the human task participants. [Figure 31–9](#) shows an example of an attachment region.

Figure 31–9 Attachment Region

31.4.1.7 Complete Task without Payload

This option creates the combination of all of the preceding task display form components (the task header, task history, custom actions, system actions, task comments, and task attachments).

31.4.1.8 Complete Task with Payload

This option creates the combination of all the preceding task display form components (the task header, task history, custom actions, system actions, task comments, and task 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 you should drag and drop the individual sections and use one of the standard ADF drop handlers, as described in [Section 31.4.2, "Standard Drop Handlers."](#)

31.4.1.9 Task Display for Notification

This option creates an ADF region that renders well when sent by e-mail. It generates the form shown in [Figure 31–10](#).

Figure 31–10

The screenshot shows an ADF form titled "Messages". At the top, there is a "Script" area containing the following code:

```

/** Shows the popup dialog */
function showCommentDialog(){
var popup = AdfPage.PAGE.findComponent("popupAddCommentDialog");
popup.show();
}

```

Below the script, the form is divided into two main sections, each enclosed in a dashed border:

Section 1: Contains several input fields for task details:

- Task Number: `#{...taskNumber.inputValue}`
- State: `#{...}`
- Outcome: `#{...outcome.inputValue}`
- Priority: `value` (with a dropdown menu)
- Creator: `#{...creator.inputValue}`
- Created Date: `#{...createdDate.inputValue}`
- Updated Date: `#{...updatedDate.inputValue}`
- Expiration Date: `#{...expirationDate.inputValue}`
- Assignees: `#{...id}[#{...}]`
- Acquired By: `#{...acquiredBy.inputValue}`

Section 2: Contains five input fields, each with a hint label:

- `#{...hints.label}` `#{...Location.inputValue}`
- `#{...hints.label}` `#{...type.inputValue}`
- `#{...hints.label}` `#{...problemDescription.inputValue}`
- `#{...hints.label}` `#{...severity.inputValue}`
- `#{...hints.label}` `#{...status.inputValue}`

Section 3: Contains five input fields, each with a hint label:

- `#{...hints.label}` `#{...ID.inputValue}`
- `#{...hints.label}` `#{...FirstName.inputValue}`
- `#{...hints.label}` `#{...LastName.inputValue}`
- `#{...hints.label}` `#{...Email.inputValue}`
- `#{...hints.label}` `#{...phone.inputValue}`

See [Section 31.5, "Creating an E-Mail Notification,"](#) for more information.

31.4.2 Standard Drop Handlers

Standard drop handlers are used for the payload section of the task details page. Depending on the element type (attribute, list, and so on) and whether the field must be editable or not, use the appropriate drop handler, as described in [Table 31–1](#).

Table 31–1 Standard ADF Drop Handlers

Drop Handler	Options
Forms	<ul style="list-style-type: none"> ADF Form—In the Edit Form Fields dialog, select individual attributes for which fields are created, or allow fields to be created for all attributes by default. You can also select the label and UI component used for each attribute. The default names for components are <code>inputText</code> and <code>selectInputDate</code>. Validation for the attributes is handled by a <code>validator</code> tag. ADF Read-Only Form—Same as the ADF form, except the default name for components is <code>outputText</code> (including for attributes of type <code>date</code>), and there is no <code>validator</code> tag.
Tables	<ul style="list-style-type: none"> ADF Table—Select the specific attributes for the table columns to display and UI components for data display. Each attribute is displayed in an <code>inputText</code> component. ADF Read-Only Table—Same as the ADF table, except each attribute is displayed in an <code>outputText</code> component. ADF Read-Only Dynamic Table—The attributes returned and displayed are determined dynamically.
Master-Details	<ul style="list-style-type: none"> ADF Master Form, Detail Form—The master and detail objects are displayed in separate forms. When a specific master data object is displayed in the top form, the first related detail data object is displayed in the form below it. ADF Master Form, Detail Table—The master object is displayed in a read-only form and the detail object is displayed in a read-only table under the form. When a specific master data object is displayed in the form, the related detail data objects are displayed in a table below it. ADF Master Table, Detail Form—The master object is displayed in a table and the detail object is displayed in a read-only form under the table. When a specific data object is selected in the master table, the first related detail data object is displayed in the form below it. ADF Master Table, Detail Table—The master and detail objects are displayed in separate tables. When a specific master data object is selected in the top table, the first set of related detail data objects are displayed in the table below it.
Single Selections	<ul style="list-style-type: none"> ADF Select One Radio ADF Select One Listbox—With the <code>selectOneListbox</code> component, create a static list of items. ADF Select One Choice—With the <code>selectOneChoice</code> component, create a static list of items or a list that is populated dynamically.
Trees	<ul style="list-style-type: none"> ADF Tree Table—A hierarchy of master-detail collections is displayed in a tree table. The advantage of using a tree table (rather than a tree) is that the tree table enables users to focus the view on a particular node in the tree. ADF Tree—A hierarchy of master-detail related objects is displayed. An ADF tree can display multiple root nodes that are populated by a binding on a master data collection. Each node in the tree can have any number of branches, which are populated by bindings on detail objects.
Navigation	<ul style="list-style-type: none"> ADF Navigation List ADF Navigation Buttons—Navigate using First, Last, Next, and Previous buttons.

See *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* for more information about standard ADF drop handlers.

31.4.3 How to Create a Task Display Form Using Individual Drop Handlers

The following steps describe how to create the task display form—a `.jspx` file—for a help desk request business process. When you finish, the task display form will have

six regions, for the custom action buttons, system action buttons, task header, comments, attachments, and task history.

See the alternative discussed in [Section 31.4.5, "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:

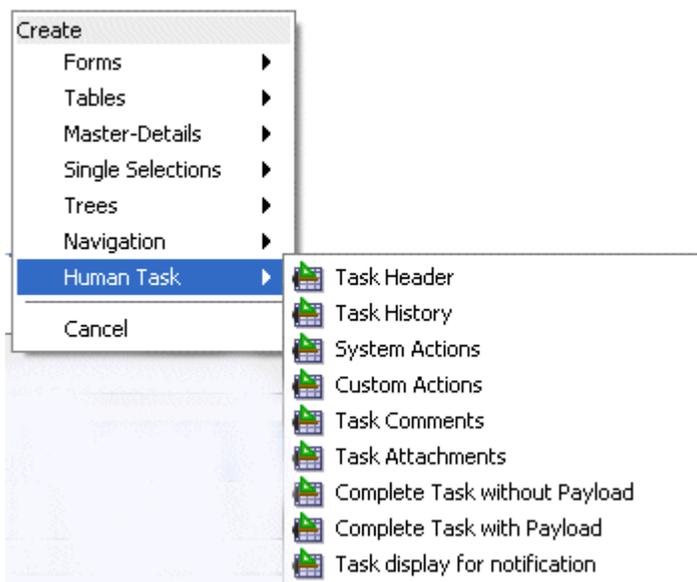
- A new application and SOA project, and a human task service—a single approver workflow in which the help desk request is either resolved or unresolved by a single help desk employee. See [Section 26.8.3, "Modeling the Help Desk Request,"](#) for more information.
- An ADF task flow based on the human task. See [Section 31.3.2, "How to Create an ADF Task Flow Within the Same Composite Application as the Human Task,"](#) for more information.

To create a task display form for the help desk request example:

Note: Click **Save All** frequently, especially after each drag and drop.

1. In the visual editor, double-click **taskDetails**.
2. In the Create JSF Page dialog, accept the defaults and click **OK**.
3. In the **Application Navigator Data Controls** panel, expand **HelpDeskRequestHumanTask**, then **loadData(String, String, String)**, and then **Return**.
4. Drag and drop the **task** icon into the **taskDetails.jspx** window.
5. Select **Human Task**, then **Custom Actions**, as shown in [Figure 31–11](#).

Figure 31–11



6. In the Edit Action Bindings dialog, accept the defaults and click **OK**.
This creates buttons for custom actions: **Resolved** and **Unresolved**, as shown in [Figure 31–12](#).

Figure 31–12

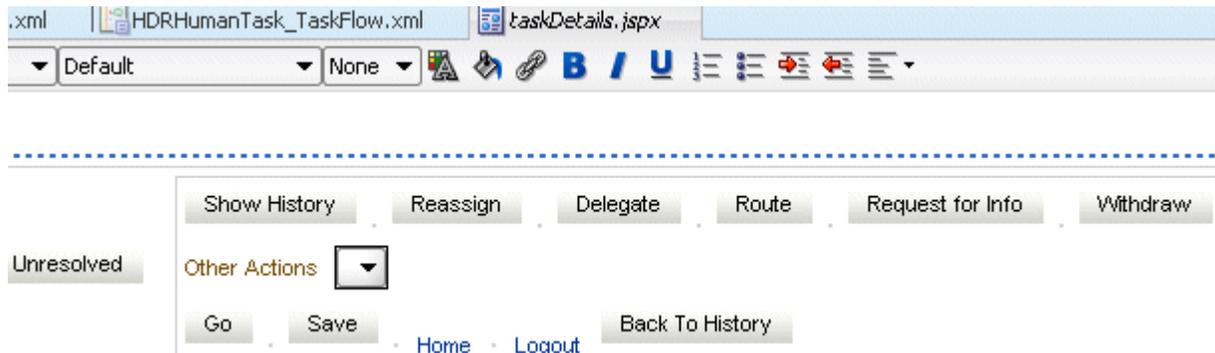


Temporary Workaround: The buttons may not be displayed due to a bug. Go to the source view, add an empty space *anywhere*, click **Save**, and return to the design view. The buttons are displayed.

7. Drag and drop the **task** icon again, this time to the immediate right of the **Unresolved** button, so that you stay inside the panel of buttons (af:panelgrouplayout).
8. Select **Human Task**, then **System Actions**.
9. In the Edit Action Bindings dialog, accept the defaults and click **OK**.

This creates buttons for system actions, as shown in Figure 31–13.

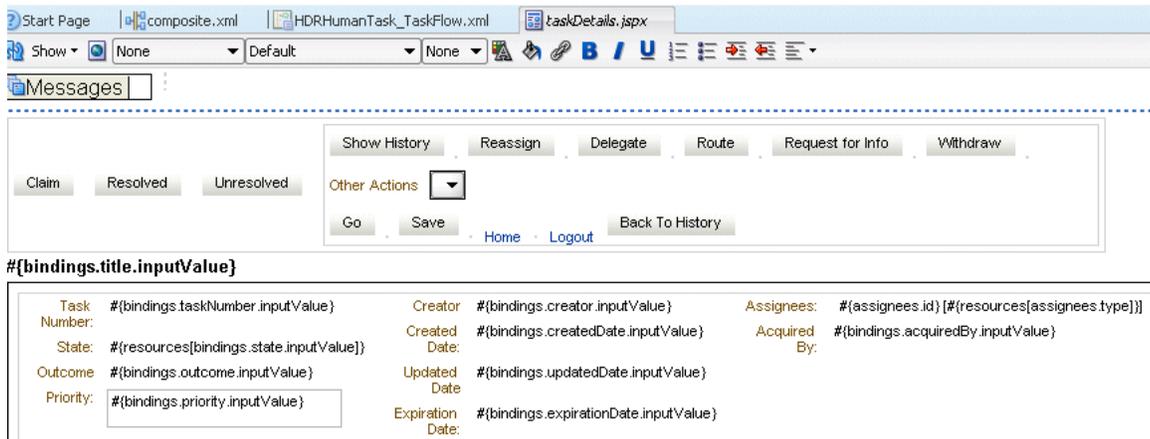
Figure 31–13



10. Drag and drop the **task** icon again, this time below the panel of buttons.
11. Select **Human Task**, then **Task Header**.

With each element that you drag and drop, the JSPX window continues to expand, as shown in Figure 31–14.

Figure 31–14



12. Drag and drop additional **task** icons into the JSPX window, selecting these options with each iteration:
 - **Human Task**, then **Task Comments**.
 - **Human Task**, then **Task Attachments**.
 - **Human Task**, then **Task History**

The task display form now has six regions, for custom action buttons, system action buttons, the task header, comments, attachments, and task history.

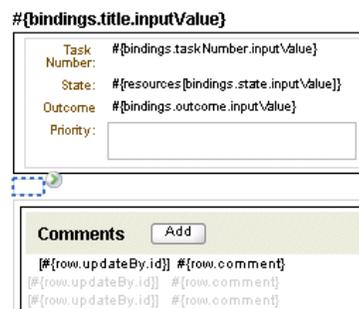
31.4.4 How to Add the Payload to the Task Display Form

Content to come.

To add the payload to the task display form:

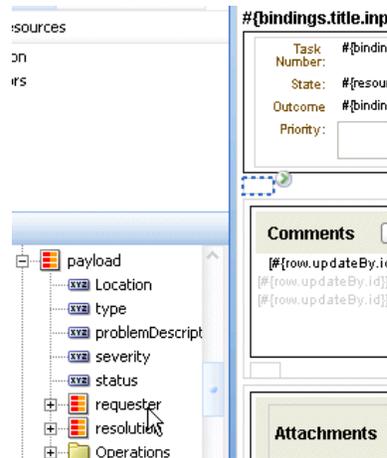
1. From the **Component Palette**, select **ADF Faces**.
2. Expand **Layout**.
3. Drag and drop **Panel Group Layout** between the **Header** and **Comment** sections, as shown in [Figure 31–15](#)

Figure 31–15



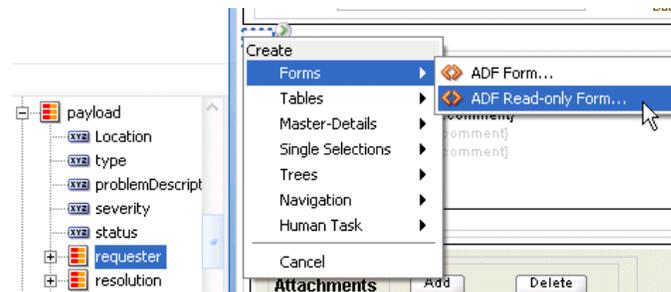
4. In the **Data Controls** panel, expand **task**, and then **payload**.
5. Drag and drop **requester** in front of the area that represents the **Panel Group Layout**, as shown in [Figure 31–16](#).

Figure 31–16



- From the context menu, select **Forms**, then **ADF Read-only Form**, as shown in Figure 31–17.

Figure 31–17



- In the Edit Form Fields dialog, accept the defaults and click **OK**.
This creates the following portion of the payload region, between the **Header** and **Comments**, as shown in Figure 31–18.

Figure 31–18

```

#{bindings.ID.hints.label}    #{bindings.ID.inputValue}
#{bindings.FirstName.hints.label}    #{bindings.FirstName.inputValue}
#{bindings.LastName.hints.label}    #{bindings.LastName.inputValue}
    #{bindings.Email.hints.label}    #{bindings.Email.inputValue}
    #{bindings.phone.hints.label}    #{bindings.phone.inputValue}
    
```

- Drag and drop **payload** below the **Panel Group Layout** and select **Forms**, then **ADF Forms**.
- In the Edit Form Fields dialog, accept the defaults and click **OK**.
- Drag and drop **resolution** below the payload region and select **Forms**, then **ADF Form**.
- In the Edit Form Fields dialog, accept the defaults and click **OK**.

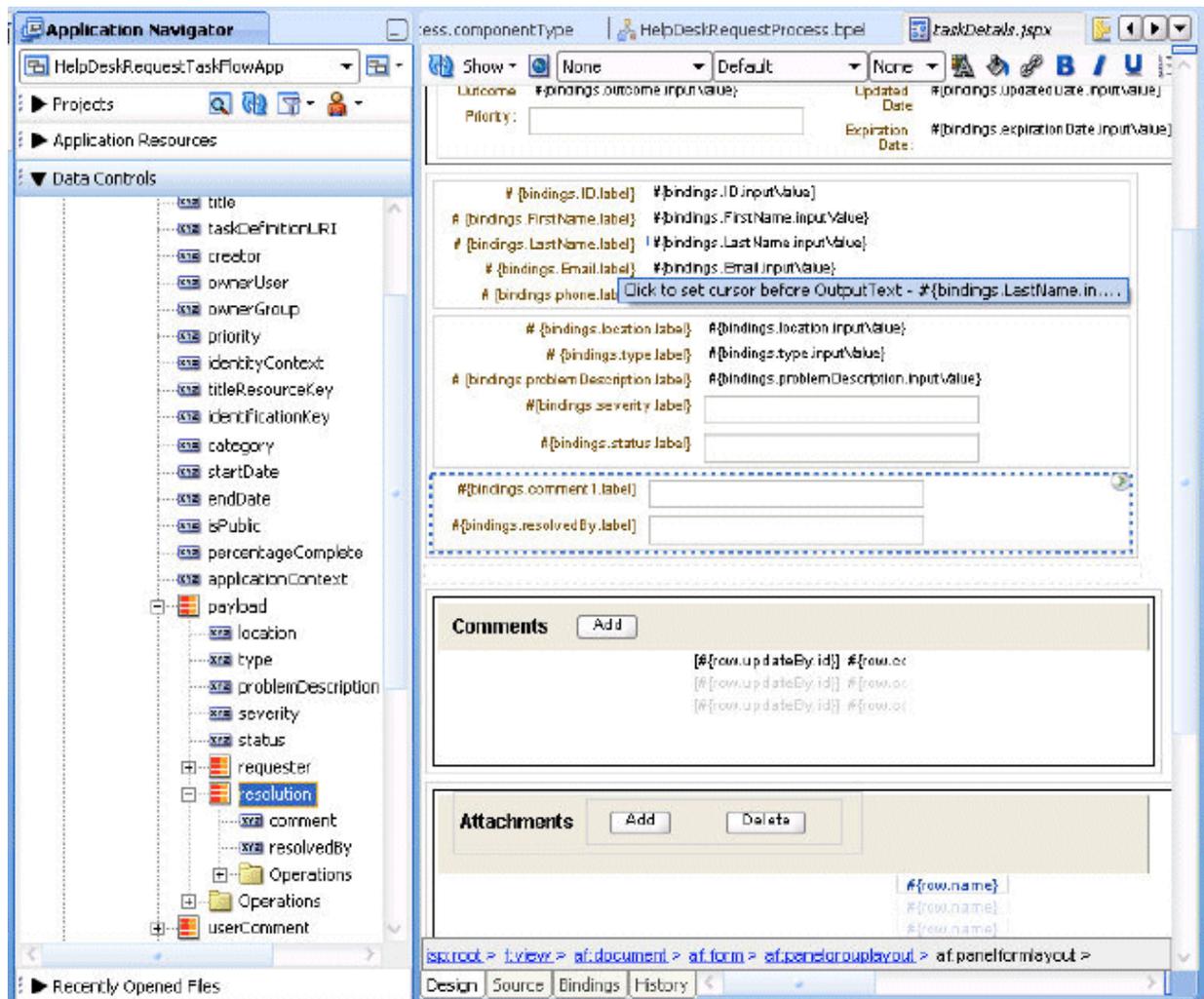
The payload regions appear, as shown in [Figure 31–19](#).

Figure 31–19

<code>#{bindings.ID.hints.label}</code>	<code>#{bindings.ID.inputValue}</code>
<code>#{bindings.FirstName.hints.label}</code>	<code>#{bindings.FirstName.inputValue}</code>
<code>#{bindings.LastName.hints.label}</code>	<code>#{bindings.LastName.inputValue}</code>
<code>#{bindings.Email.hints.label}</code>	<code>#{bindings.Email.inputValue}</code>
<code>#{bindings.phone.hints.label}</code>	<code>#{bindings.phone.inputValue}</code>
<code>#{bindings.Location.hints.label}</code>	<code>#{bindings.Location.inputValue}</code>
<code>#{bindings.type.hints.label}</code>	<code>#{bindings.type.inputValue}</code>
<code>#{bindings.problemDescription.hints.label}</code>	<code>#{bindings.problemDescription.inputValue}</code>
<code>#{bindings.severity.hints.label}</code>	<code>#{bindings.severity.inputValue}</code>
<code>#{bindings.status.hints.label}</code>	<code>#{bindings.status.inputValue}</code>
<code>#{bindings.comment1.hints.label}</code>	<code>#{bindings.comment1.inputValue}</code>
<code>#{bindings.resolvedBy.hints.label}</code>	<code>#{bindings.resolvedBy.inputValue}</code>

The task display form, shown in [Figure 31–20](#), is completed and ready to be deployed.

Figure 31–20 The Task Display Form (taskDetails.jspx)



31.4.5 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 having to individually build each region.

See the alternative discussed in [Section 31.4.3, "How to Create a Task Display Form Using Individual Drop Handlers."](#)

To create a task display form using the Complete Task with Payload drop handler:

Note: Click **Save All** frequently, especially after each drag and drop.

1. In the visual editor, double-click **taskDetails**.
2. In the Create JSF Page dialog, accept the defaults and click **OK**.

3. In the **Application Navigator Data Controls** panel, expand the human task name, then **loadData(String, String, String)**, and then **Return**.
4. Drag and drop the **task** icon into the **taskDetails.jspx** window.
5. Select **Human Task**, then **Complete Task with Payload**.
6. In the Edit Action Bindings dialog, select a data collection and the action you want the data control to initiate and click **OK**.
- 7.
- 8.

You are ready to add the payload region to the task display form.

31.4.6 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 31.6, "Deploying a Composite Application with a Task Flow,"](#) for more information.

31.5 Creating an E-Mail Notification

A task display form is used to provide an e-mail notification if e-mail notification is defined as part of the human task. You have the following options for an e-mail notification:

- **Default e-mail 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 e-mail. Images in the task flow are included as attachments so that the notification can be viewed in disconnected mode.
- **Custom e-mail notification**—Create an e-mail notification task page that is different from the task display form used in the worklist. When creating this e-mail notification page, you use the same human task-aware data controls that you use for other task display forms.

See [Section 28.2, "Notifications from Human Workflow,"](#) to review notification settings as part of a human task definition (`.task` file).

31.5.1 How to Create an E-Mail Notification

To send a custom e-mail notification whose content and layout you have specified (instead of sending the default page), you create another JSPX file in which you design an e-mail notification page. You can create the notification page from scratch by using the custom and standard drop handlers, or you can use the e-mail notification drop handler. In addition, do the following:

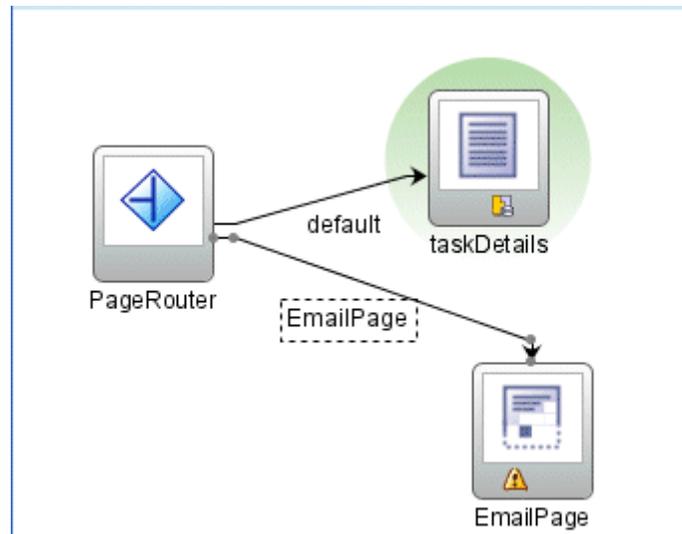
- Add a router to the task flow. The router directs the task flow to send either the e-mail notification page or the default page, depending on the control flow based on the `bpmClientType` page flow scope value.
- Specify inline styles to use for the layout, because a cascading style sheet (CSS) is not attached to the e-mail and the default ADF CSS is not included

- 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.)

31.5.1.1 Creating a Task Flow with a Router

The control flow case with a router lets you direct the request to a specific page based on certain parameters. (Figure 31–21 shows the router directing the request to an e-mail page.) For an ADF task flow based on a human task, you may need a special page for e-mail notifications or for digital signatures. This section describes how to create a special page for e-mail notifications.

Figure 31–21 Task Flow with a Router



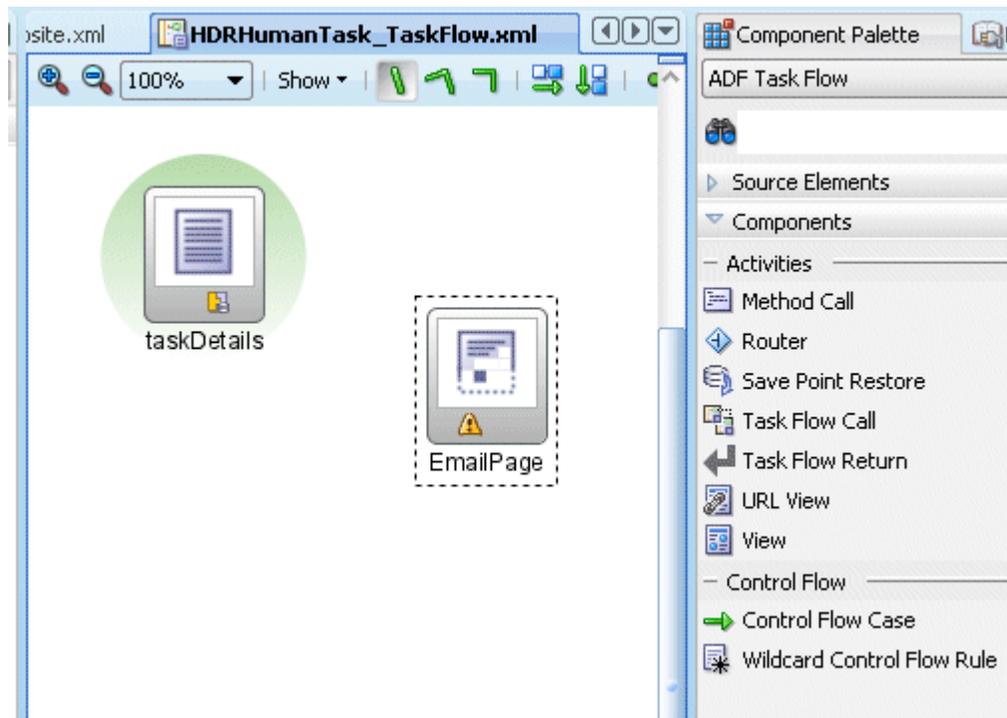
To create a task flow with a router:

1. In the **Application Navigator**, expand the **HelpDeskRequestFlow** project and double-click **HelpDeskRequestHumanTask_TaskFlow**.

The `HelpDeskRequestHumanTask_TaskFlow.xml` file opens in the visual editor. In the diagram view, you see the **taskDetails** icon.

2. From the **Component Palette**, drag and drop the **View** icon into the visual editor.
3. Click **view1** below the icon and enter `EmailPage`, as shown in Figure 31–22

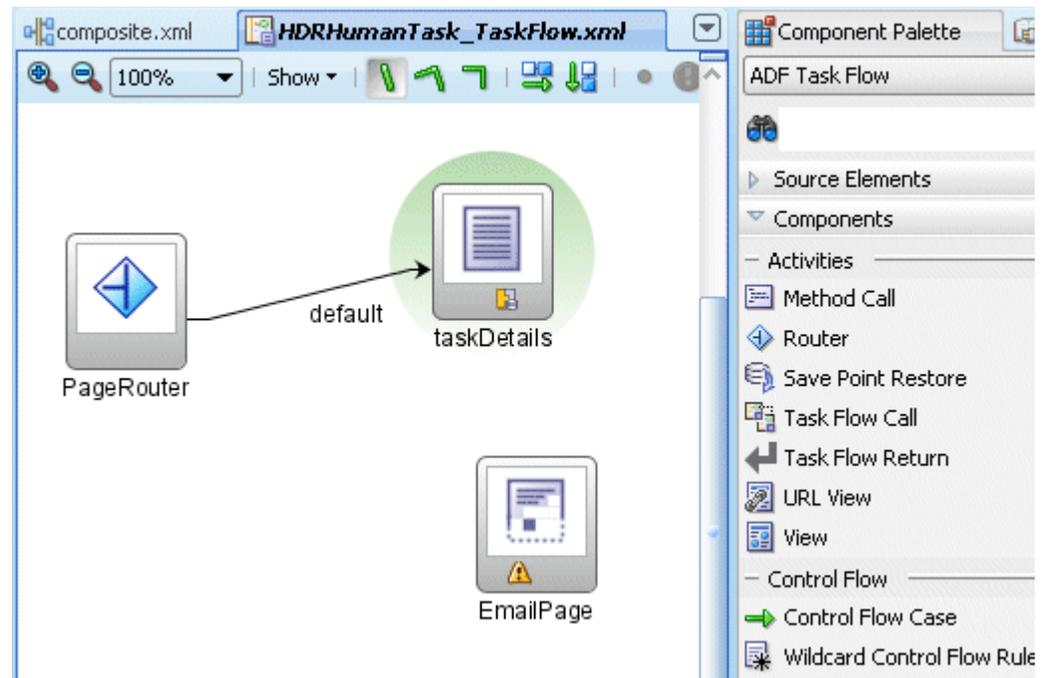
Figure 31–22



4. From the **Component Palette**, drag and drop the **Router** icon into the visual editor.
5. Click **router1** below the icon and enter `PageRouter`.
6. Click the **router - PageRouter - Property Inspector** tab.
7. In the **default-outcome** field, enter `default`.
8. Add more cases.
9. In the **outcome** field, enter `EmailPage`.
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 visual editor, drag a control flow from **PageRouter** to **taskDetails**.

The control flow is automatically labeled **default**, as shown in [Figure 31–23](#).

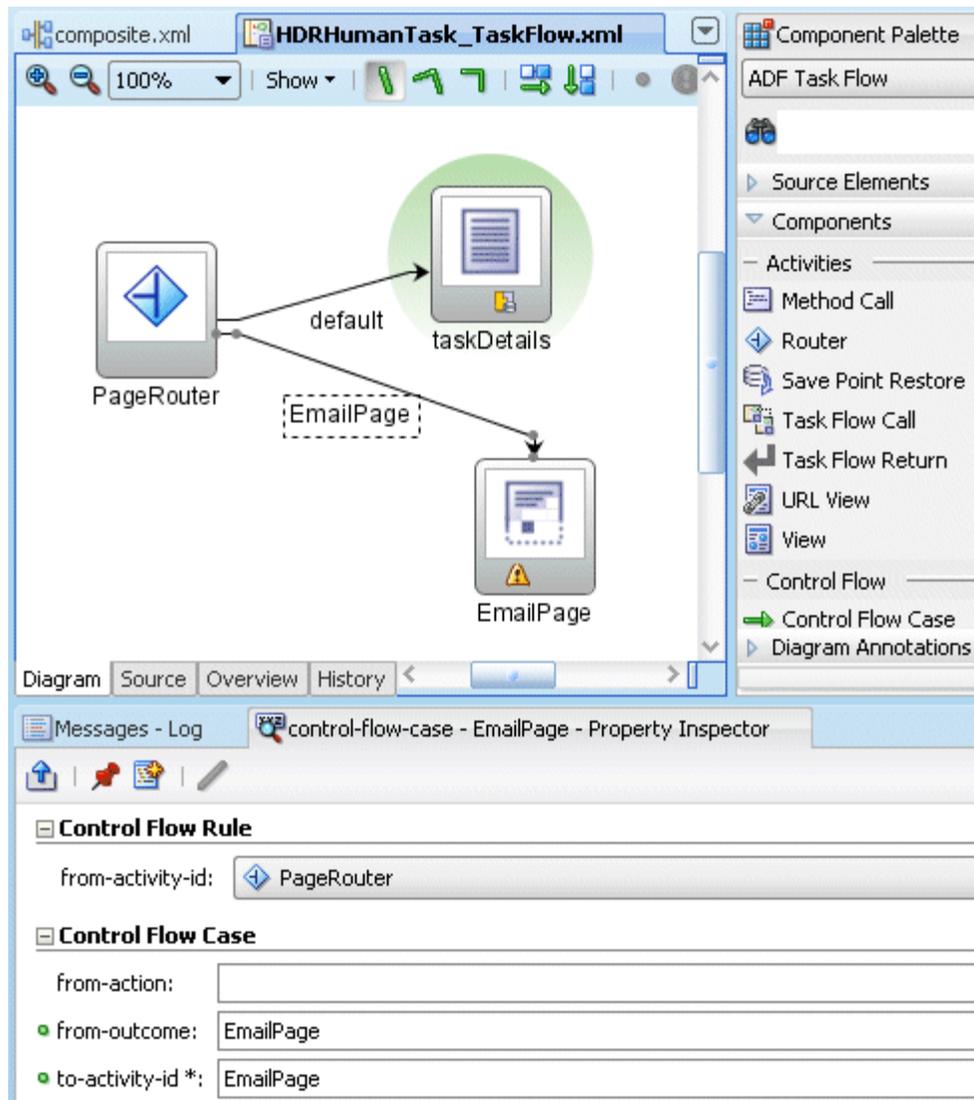
Figure 31–23



13. In the **Component Palette**, click **Control Flow Case**.
14. In the visual editor, drag a control flow from **PageRouter** to **EmailPage**.
15. Click the control flow to see the **control-flow-case - EmailPage - Property Inspector**.
16. In the **from-outcome** field, enter EmailPage.

[Figure 31–24](#) shows the completed control flow.

Figure 31–24



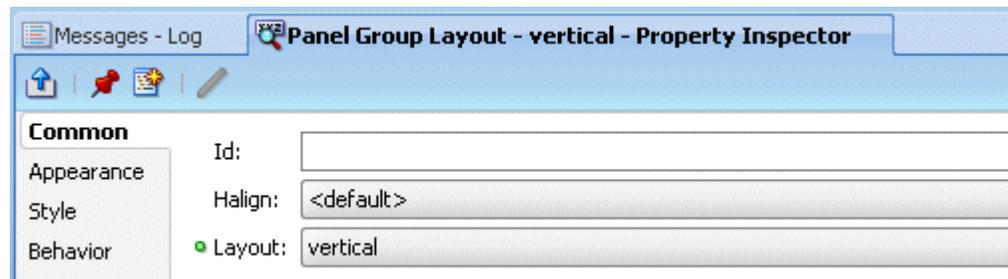
31.5.1.2 Creating an E-Mail Notification Page

Creating an e-mail notification page is similar to creating a task display form, with the addition of defining inline styles.

To create an e-mail notification page:

1. In the visual editor, double-click **EmailPage**.
2. In the Create JSF Page dialog, accept the defaults and click **OK**.
The **EmailPage.jspx** tab opens in the visual editor.
3. From the **Component Palette**, drag and drop **Panel Group Layout** into the visual editor.
4. In the **Panel Group Layout - vertical - Property Inspector** tab, from the **Layout** list, select **vertical**, as shown in [Figure 31–25](#).

Figure 31–25



5. Click **Style**, and, in the **InlineStyle** field, enter the following:
margin: 2px; border: 1px solid black; padding: 3px; overflow: auto; width: 95%
6. From the **Component Palette**, drag and drop **Image** into the panel group layout area.
7. In the Insert Image dialog, search for an image, in this example, `oracle.gif`.
If you are asked if you want to put the image in the current document root, click **Yes**.
8. Click **Finish**.
9. From the **Data Controls** panel, expand **HelpDeskRequestHumanTask**, then **loadData(String, String, String)**, and then **Return**.
10. Drag and drop **task** into the panel group layout area.
11. Select **Human Task**, and then **Task Header**.
12. In the Edit Action Binding dialog, accept the defaults and click **OK**.
13. Drag and drop additional **task** icons into the JSPX window, selecting these options with each iteration:
 - **Human Task**, then **Task Comments**.
 - **Human Task**, then **Task Attachments**.
 - **Human Task**, then **Task History**

To continue adding the task payload, see Step 1 in [Section 31.4.4, "How to Add the Payload to the Task Display Form."](#)

31.5.2 What Happens When You Create an E-Mail Notification Page

The e-mail notification page is sent as HTML content in the e-mail message body. Images on the page are inlined as attachments. Relative URLs are converted to absolute URLs.

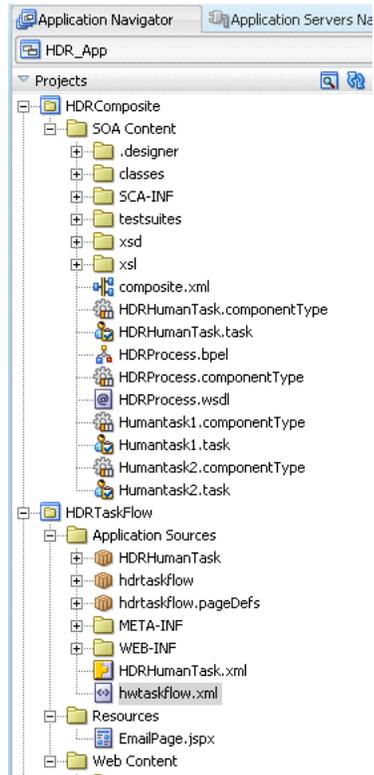
31.6 Deploying a Composite Application with a Task Flow

The composite application containing the task flow must be deployed to the application server as a Web archive (WAR) module before you can use the task display form in the Worklist Application. The WAR module is created as an application assembly in an Oracle archive (OAR) module when you create the task flow project.

31.6.1 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 31–26](#) shows the file in the Application Navigator.

Figure 31–26 The `hwtaskflow.xml` File



[Example 31–1](#) shows a sample `hwtaskflow.xml` file with comments on which values can and cannot be changed.

Example 31–1 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/>
</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></hostname>
  <httpPort></httpPort>
  <httpsPort></httpsPort>
</TaskFlowServer>

<!-- Miscellaneous configurable properties for Task Flow -->
<Property name = "" value = ""/>

<!-- Task Flow specific properties -->
<hwTaskFlow>
  <WorkflowName></WorkflowName>
  <TaskDefinitionNamespace></TaskDefinitionNamespace>
  <TaskFlowId></TaskFlowId>
  <TaskFlowFileName></TaskFlowFileName>
</hwTaskFlow>

</hwTaskFlows>

```

31.6.2 How to Deploy a Composite Application with a Task Flow

The following instructions require an application server connection.

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*.

The Configure Application dialog appears.

If you do not have a connection, select **New Connection** and use the Application Server Connection wizard.

2. In **New Revision ID**, enter 1 . 0.
3. Click **OK**.

If you are using a local EJB, then do the following:

- a. Edit the `LookupType` value in `hwtaskflow.xml` as follows:

```
<LookupType>LOCAL</LookupType>
```

See [Section 31.6.1, "Before Deploying the Task Display Form: Port Changes,"](#) for more information.

- b. Select `soa-infra` as the **parent app** during deployment.

31.6.3 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.

31.6.4 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.

To deploy a task flow as a separate application:

1. Select **New > Application** from the **File** menu.
2. Enter **HelpDeskRequestTaskFlowApp** in the **Application Name** field.
3. Accept the default setting for all other fields.
4. Click **OK**.
5. Click **Cancel** on the Create Project window.
6. Select **Save All** from the **File** main menu.
7. Select the **Application Menu** list for the application you just created.
8. Select **New Project** from the list that appears.
The New Gallery window appears.
9. Select **Empty Project** from the **Items** list.
10. Click **OK**.
11. Enter **HelpDeskRequestTaskFlow** in the **Project Name** field, and click **OK**.
12. Right-click the composite application name, select **Deploy**, and then *application_name* > *to* > *application_server_connection*.
The Configure Application dialog appears.
If you do not have a connection, select **New Connection** and use the Application Server Connection wizard.
13. In **New Revision ID**, enter 1 . 0.
14. Click **OK**.
If you are using a local EJB, then do the following:
 - a. Edit the `LookupType` value in `hwtaskflow.xml` as follows:

```
<LookupType>LOCAL</LookupType>
```

See [Section 31.6.1, "Before Deploying the Task Display Form: Port Changes,"](#) for more information.
 - b. Select **soa-infra** as the **parent app** during deployment.
15. Right-click **HelpDeskRequestTaskFlowApp**, select **Deploy**, and then **HelpDeskRequestTaskFlowApp** module to **SOAConnection**.
16. Click **OK**.
If you are using a local EJB, then do the following:
 - a. Edit the `LookupType` value in `hwtaskflow.xml` as follows:

```
<LookupType>LOCAL</LookupType>
```

See [Section 31.6.1, "Before Deploying the Task Display Form: Port Changes,"](#) for more information.

- b. Select `soa-infra` as the **parent app** during deployment.

31.6.5 What Happens When You Deploy the Task Display Form

After deployment, the task display form is available for display in the worklist. Users log in to the worklist to act on their tasks.

See the following for more information:

- [Section 31.7.1, "How to Display the Task Display Form in the Worklist,"](#) for how to log in
- [Chapter 33, "Using Oracle BPM Worklist,"](#) for how to act on tasks

31.7 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 31–27](#) shows how the task display form for the help desk request example is displayed in the worklist task details page.

Figure 31–27 Worklist Task Details Page

Resolved Unresolved Show History Reassign Route Other Actions Go »

Help desk request wfault

Task Number: 200001	Creator	Assignees: jstein [U]
State: Assigned	Created Date: 4/12/07 4:42:44 PM	Acquired By:
Outcome	Updated Date: 4/12/07 4:42:44 PM	
Priority: <input style="width: 100px;" type="text" value="3"/>	Expiration Date: 4/13/07 4:52:44 PM	

ID wfault

FirstName William

LastName Faulkner

Email user1@us.oracle.com

Phone

Location California

Type Hardware

ProblemDescription Unable to reboot the system

Severity

Status

Comment

ResolvedBy

Comments

Attachments

Short History

Version	Action	State:	Outcome
1	TASK_VERSION_REASON_INITIATED	ASSIGNED	

See [Section 33.4, "Acting on Tasks: The Task Details Page,"](#) for more information.

31.7.1 How to Display the Task Display Form in the Worklist

Use Internet Explorer 7 or Firefox 2.0.0.2 to access the worklist.

To log in to the worklist:

1. Go to

`http://host_name:port_number/integration/worklistapp`

- *host_name* is the name of the host on which Oracle BPEL Process Manager is installed
- The *port_number* used at installation (typically 8888) is noted in `SOA_Oracle_Home\install\bpelsetupinfo.text`

2. Enter the username and password.

You can use `jstein` and `welcome1`, or `oc4jadmin` and `welcome1` to log in as an administrator.

The user name and password must exist in the user community provided to JAZN.

3. Select a realm.
4. Click **Login**.

31.8 Troubleshooting the Task Display Form

If you are unable to see the human task details when you click on the task in the Worklist Application, then use the following steps to troubleshoot the problem.

- Design-time troubleshooting: Ensure that you start JDeveloper using `jdev.exe` instead of `jdevw.exe`. This will bring up a console window in the background that shows any error messages or stack traces.
- Deployment issues: If you see any errors during deployment of the ADF task flow, check the following:
 - Note that you must deploy the "ADF task flow based on human task" on the same instance as the SOA server. Additionally, make sure you specify "soa-infra" as the parent application when you deploy this ADF task flow application
 - Check if the `taskflow.properties` file exists in your project and has the following settings:


```
human.task.lookup.type=LOCAL
```
 - Ensure that the ADF run-time libraries are installed on the OC4J instance where the task flow is deployed. You can verify this by checking if the `adf.oracle.domain` shared library exists in `Oracle_Home/j2ee/home/config/server.xml`
- Run-time issues: If you see the task in the task list but you get an error when clicking on the task title, use the following steps to troubleshoot the problem:
 - Check if you had any errors during deployment as described above.
 - Enable logging for ADF as described in the "logging" section below. Set the log level to FINE and deploy the task flow again. After this you should see error messages in the `OC4J_HOME/j2ee/home/log/oc4j/log.xml` file.
 - If you get errors on the task details screen when performing any operations, check the following log files:

```
OC4J_HOME/j2ee/home/log/oc4j/log.xml
OC4J_HOME/j2ee/home/log/server.log
OC4J_HOME/j2ee/home/application-deployments/soa-infra/application.log
OC4J_HOME/oc4j.log
```

ADF task flow logging: To enable ADF logging, add following fragments in `OC4J_HOME/j2ee/home/config/j2ee-logging.xml` and restart the server.

Example 31-2 Enabling ADF Logging

```
<logger name='oracle.adf' level='FINE' useParentHandlers='false'>
  <handler name='oc4j-handler' />
```

```
        <handler name='console-handler' />
    </logger>
<logger name='oracle.adfinternal' level='FINE' useParentHandlers='false'>
    <handler name='oc4j-handler' />
    <handler name='console-handler' />
</logger>
<logger name='oracle.jbo' level='FINE' useParentHandlers='false'>
    <handler name='oc4j-handler' />
    <handler name='console-handler' />
</logger>
```

Human Task Services

Human task services and functions are responsible for a variety of tasks. This chapter describes the human task services.

This chapter contains the following topics:

- [Section 28.1, "Human Task Services"](#)
- [Section 28.2, "Notifications from Human Workflow"](#)
- [Section 28.3, "Configuring the Assignment Service"](#)
- [Section 28.4, "Human Task Service and Identity Service Related XPath Extension Functions"](#)
- [Section 28.5, "NLS Configuration"](#)
- [Section 28.6, "Changes to APIs"](#)
- [Section 28.7, "Summary"](#)

28.1 Human Task Services

This section describes the responsibilities of the following human task services:

- [Section 28.1.1, "EJB, SOAP, and Java Support for the Human Task Services"](#)
- [Section 28.1.2, "Security Model for Services"](#)
- [Section 28.1.3, "Task Service"](#)
- [Section 28.1.4, "Task Query Service"](#)
- [Section 28.1.5, "Identity Service"](#)
- [Section 28.1.6, "Notification Service"](#)
- [Section 28.1.7, "Task Metadata Service"](#)
- [Section 28.1.8, "User Metadata Service"](#)
- [Section 28.1.9, "Runtime Config Service"](#)
- [Section 28.1.10, "Digital Signatures and the Evidence Store Service"](#)

See Also: [Section 26.3, "Workflow Services Components"](#) on page 26-8

28.1.1 EJB, SOAP, and Java Support for the Human Task Services

[Table 28-1](#) lists the type of SOAP, EJB, and Java support provided for the task services. Most human task services are accessible through SOAP and local and remote EJB 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 28–1 EJB, SOAP, and Java Support

Service Name	Supports SOAP Web Services	Supports Remote EJB	Supports Local EJB
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	No
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 run time environment.	Yes	Yes	Yes
Evidence Store Service — Supports storage and nonrepudiation of digitally-signed workflow tasks	Yes	Yes	Yes
Identity Service (BPM authentication and authorization services) — Enables authentication and authorization of users and the lookup of user properties, roles, group memberships, and privileges.	Yes	No	No

Table 28–2 lists the location for the SOAP WSDL file for each task service.

Table 28–2 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>
Notification Service	<code>http://host:port/integration/services/NotificationService/NotificationService?WSDL</code>

Table 28–2 (Cont.) SOAP WSDL Location for the Task Services

Service name	SOAP WSDL location
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>
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>

See Also: [Appendix A, "Building a Custom Worklist Client"](#) for details about the client library for worklist services

28.1.2 Security Model for Services

With the exception of IDM, all services that use the above-mentioned APIs (SOAP, remote EJB, local EJB, and Java WSIF) require authentication to be invoked. All the above channels support passing the user identity using the human task context. The human task 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 task context used for all services. Optionally, with each request, you can pass the human task 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 enables you to create the context for a logged-in user to the Oracle BPEL Worklist Application if the password for that user is not available.

28.1.2.1 Limitation on Propagating Identity to Workflow Services when Using SOAP Web Services

SOAP Web services also support Web service security. When Web service security is used, the human task context does not need to be present in the SOAP input. The Web service security can be configured from the Oracle Enterprise Manager 11g Application Server Control Console.

Note: Human task service SOAP clients cannot be used when Web service security is used.

See Also: "Configuring Single Sign-on Using SAML" in the *Oracle Application Server Web Services Security Guide* for details about propagating the identity of a user from a Web application to the Web service

28.1.2.2 Security in EJBs

If EJB identity propagation is enabled, the name of the user currently authenticated by the EJB container is automatically propagated to the various workflow services. A null

value can be specified for `IWorkflowContext` in workflow service methods in this case.

These steps provide a high-level overview of how to enable identity propagation.

1. Enable subject propagation.
 - Modify `startorappel.sh` to include the `-Dsubject.propagation=true` on both the server and client OC4Js.
 - Set `jaas-mode="doAs"` or `jaas-mode="doAsPrivileged"` in the `orion-application.xml` file of the client web application.
2. Set the subject propagation restrictions.
3. Make the `java.security.Principal` implementation available in the client application.
4. If there are multiple realms, you must set the property `jaas.username.simple=true`.

See Also: *Oracle Containers for J2EE Security Guide* for configuration instructions

28.1.2.3 Creating Human Task Context on Behalf of a User

The `authenticate` API operation on the task query service can create the human task 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.

In this example, the human task context is created for user `jcooper`.

```
ITaskQueryService taskQueryService = ...;
String realm = ...;
IWorkflowContext wfCtx =
    taskQueryService.authenticate('bpeladmin', 'welcome1', realm, 'jcooper');
```

28.1.3 Task Service

The task service exposes operations to act on tasks. [Table 28–3](#) describes the operations of the task service. Package `oracle.bpel.services.workflow.task` corresponds to the task service.

Table 28–3 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.
<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.

Table 28–3 (Cont.) Task Service Methods

Method	Description
<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 exists 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
<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.

Table 28–3 (Cont.) Task Service Methods

Method	Description
<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 task 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 adhoc 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 task permits adhoc 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>	Allows task owners (or users with the <code>BPMWorkflowSuspend</code> privilege) to put a human task 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 adhoc 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 task permits adhoc routing of the task.
<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 task. 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> .

Table 28–3 (Cont.) Task Service Methods

Method	Description
<code>withdrawTasks</code>	Withdraw a set of tasks.

See Also: *Oracle BPEL Process Manager Workflow Services API Reference* located in the `SOA_Oracle_Home\bpel\docs\workflow` directory

28.1.4 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 28–4](#) 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 28–4 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. Authentication can optionally be made on behalf of another user.
<code>createContext</code>	Creates a valid <code>IWorkflowContext</code> object from a preauthenticated HTTP request.
<code>getWorkflowContext</code>	Gets a human task context with the specified context token.
<code>destroyWorkflowContext</code>	Cleans up a human task 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.
<code>getTaskVersionDetails</code>	Gets the specific task version details for the specified task ID and version number.

Table 28–4 (Cont.) Task Query Service Methods

Method	Description
queryTasks	<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 where 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 where 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 where 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. <p>Note: To use the task query service over SOAP, call <code>Predicate.enableXMLSerialization(true)</code>; to make the predicate object serializable.</p>
queryViewTasks	<p>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.</p>

See Also: *Oracle BPEL Process Manager Workflow Services API Reference* in the documentation library

28.1.5 Identity Service

The identity service is a thin Web service layer on top of the Oracle Application Server 11g security infrastructure, namely OracleAS Identity Management (IDM) and Java

Platform Security (JPS), or any custom user repository. The identity service enables authentication and authorization of users and the lookup of user properties, roles, group memberships, and privileges. IDM is the sole identity service provider for Oracle Application Server 11g. IDM handles all storage and retrieval of users and roles for various repositories, including XML, LDAP, and so on. More, specifically IDM provides the following features:

- All providers are consolidated under IDM. 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 IDM, through configuring Oracle Virtual Directory (OVD) as an LDAP provider to IDM, or through both. OVD aggregates data across various repositories.
- Application roles are supported through the JPS policy store and groups are supported through the IDM identity store. The JSP layer handles all security-related functionality at the OC4J container level. The policy store of JPS handles all policies defined at the application level. Properties related to IDM are stored in the `jps-config.xml` file. The enterprise roles (groups) and application roles of previous releases are no longer supported.
- 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 JPS 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.

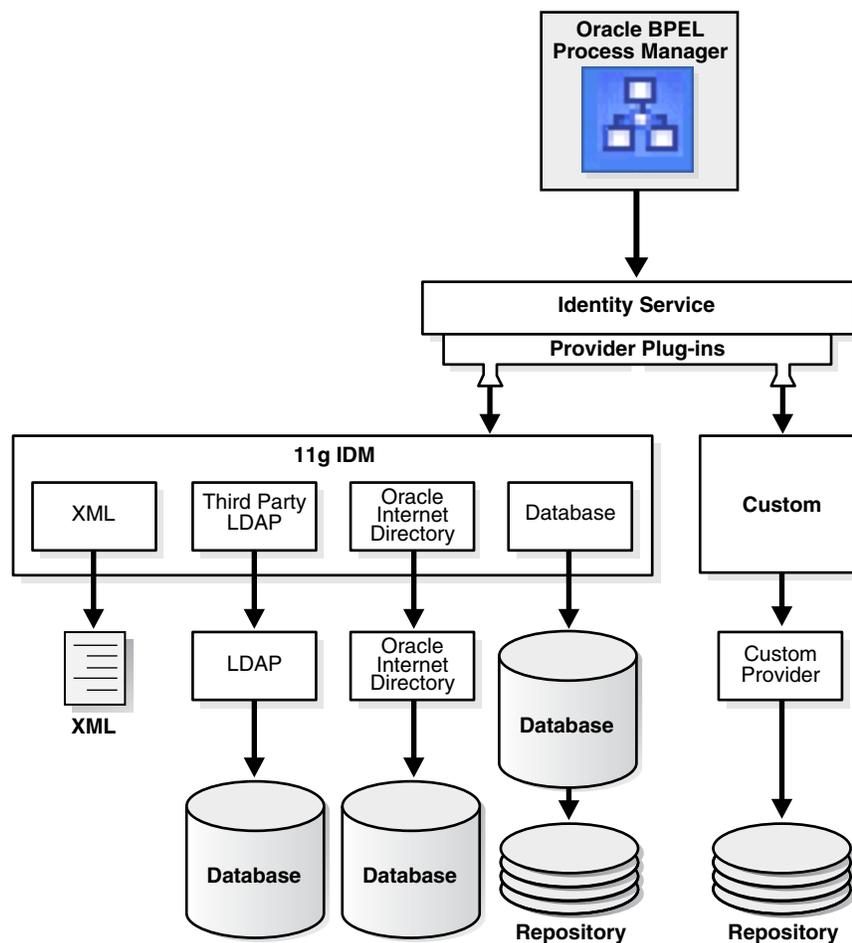
See Also:

- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details about configuring the identity service
- *Oracle Fusion Middleware Security Guide* and *Oracle Fusion Middleware Application Security Developer's Guide* for details about JPS
- *Oracle Identity and Access Management Application Developer's Guide* for details about IDM
- *Oracle Fusion Middleware Administrator's Guide for Oracle Virtual Directory* for details about OVD

28.1.5.1 Identity Service Providers

IDM is the only supported provider for release 11g, as shown in [Figure 28-1](#).

Figure 28-1 Identity Service Providers



28.1.5.1.1 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`.

See Also:

- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for configuration instructions
- `SOA_ORACLE_HOME\bpel\docs\workflow\oracle\tip\pc\services\identity` for Javadoc on the `BPMIdentityService` interface

28.1.6 Notification Service

The notification service exposes operations that can be invoked from the BPEL business process to send notifications through e-mail, voice, instant messaging (IM), or short message service (SMS) channels.

See Also:

- [Section 28.2, "Notifications from Human Workflow"](#) on page 28-20 for specific details about the notification service
- [Chapter 23, "Notifications and the Oracle User Messaging Service"](#)
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for instructions on configuring notification service delivery channels

28.1.7 Task Metadata Service

Task metadata service exposes operations to retrieve metadata information related to a task. [Table 28-5](#) describes these methods. Package `oracle.bpel.services.workflow.metadata` corresponds to the task metadata service.

Table 28-5 Task Metadata Service Methods

Method	Description
<code>getOutcomes</code>	Get the permitted outcomes of a task. The outcomes are returned with their display values.
<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>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>getTaskDisplayRegion</code>	Get the task display region for a task.
<code>getVersionTrackedAttributes</code>	Get the task attributes that when changed causes a task version creation.
<code>listTaskMetadata</code>	List the task definitions in the system.

See Also: *Oracle BPEL Process Manager Workflow Services API Reference* located in the `SOA_ORACLE_HOME\bpel\docs\workflow` directory

28.1.8 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 task 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 task 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 28–6](#) describes the methods in the user metadata service. Package `oracle.bpel.services.workflow.user` corresponds to the user metadata service.

Table 28–6 User Metadata Service Methods

Method	Description
<code>setVacationInfo</code>	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.)
<code>getVacationInfo</code>	Retrieves the date range (if any) during which a user is unavailable for the assignment of tasks.
<code>getRuleList</code>	Retrieves a list of rules for a particular user or group.
<code>getRuleDetail</code>	Gets the details for a particular human task rule.
<code>createRule</code>	Creates a new rule.
<code>updateRule</code>	Updates an existing rule.
<code>deleteRule</code>	Deletes a 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 already has the highest priority.
<code>decreaseRulePriority</code>	Decreases the priority of a rule by one. This method does nothing if this rule already has the lowest priority.
<code>getRuleSetInfo</code>	Returns information relating to the Oracle Business Rules rule set being used to store the rules for a particular user or group. This is useful if a client wants to make use of the rules SDK directly for manipulating rules, rather than using the user metadata service.
<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 task service, and are available to all users.

Table 28–6 (Cont.) User Metadata Service Methods

Method	Description
<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 the worklist application), 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.

See Also:

- [Chapter 33, "Using Oracle BPM Worklist"](#) for details about the rule configuration and user preference pages
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details on how to designate a user as a group owner
- *Oracle BPEL Process Manager Workflow Services API Reference* located in the `SOA_ORACLE_HOME\bpel\docs\workflow` directory

28.1.9 Runtime Config Service

The runtime config service provides methods for managing metadata used in the task service run time 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 a number of 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 task 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 the worklist application. 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 `TextAttribute10`
- `FormAttribute1` through `FormAttribute5`
- `UrlAttribute1` through `UrlAttribute5`
- `DateAttribute1` through `DateAttribute5`
- `NumberAttribute1` through `NumberAttribute5`

Values for protected flex fields are as follows:

- `ProtectedTextAttribute1` through `ProtectedTextAttribute10`
- `ProtectedFormAttribute1` through `ProtectedFormAttribute5`
- `ProtectedUrlAttribute1` through `ProtectedUrlAttribute5`
- `ProtectedDateAttribute1` through `ProtectedDateAttribute5`
- `ProtectedNumberAttribute1` through `ProtectedNumberAttribute5`

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 methods for querying the dynamic assignment functions supported by the server.

[Table 28–7](#) describes the methods in the runtime config service. Package `oracle.bpel.services.workflow.runtimeconfig` corresponds to the runtime config service.

Table 28–7 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>getAttributeLabelUsages</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 task 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.

See Also:

- [Section 28.3.1, "Dynamic Assignment Functions"](#) on page 28-30 for additional details
- [Chapter 33, "Using Oracle BPM Worklist"](#) for details about flex field mapping
- *Oracle BPEL Process Manager Workflow Services API Reference* located in the `SOA_ORACLE_HOME\bpel\docs\workflow` directory

28.1.9.1 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 achieve this, you can add entries to the `WorkflowLabels.properties` resource property file, and associated resource bundles in other languages. This file exists in the `SOA_ORACLE_HOME\admin\template\bpel\services\wfresource` directory.

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.

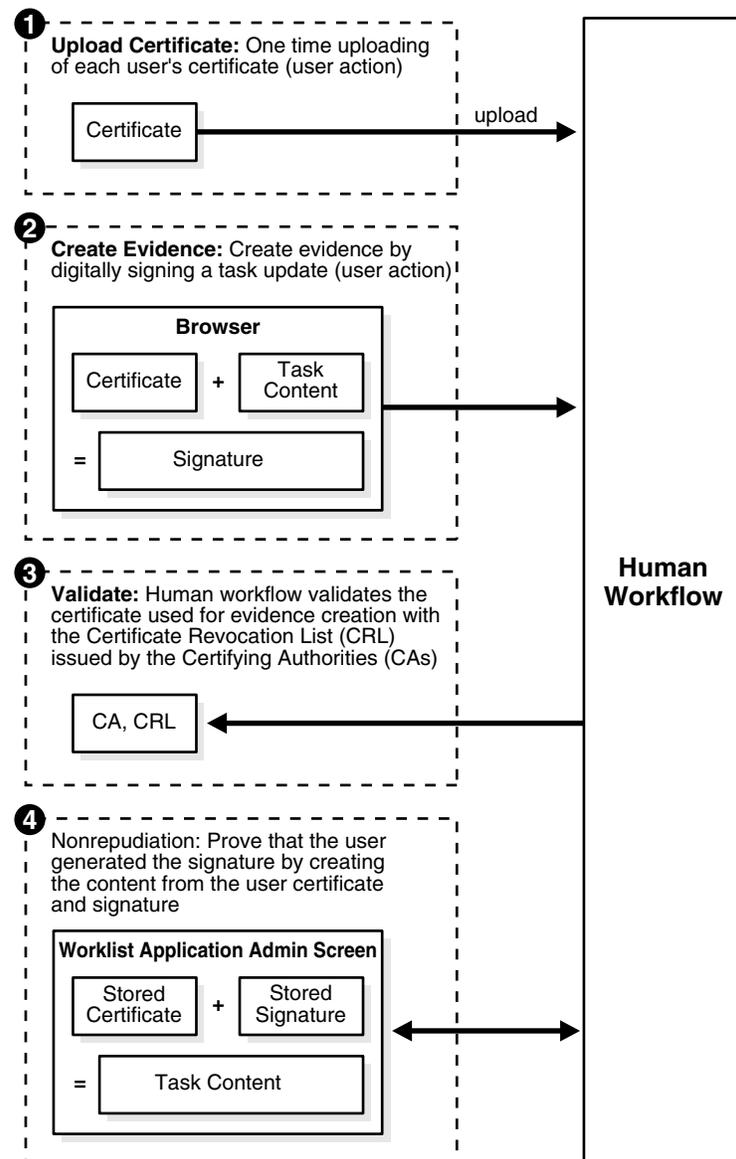
28.1.10 Digital Signatures and the Evidence Store Service

The evidence store service is used for digital signature storage and nonrepudiation of digitally-signed human tasks. 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 task 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 Worklist Application, when a user approves or rejects the task, the Web browser:

- Asks the user to choose the certificate to use for signing
- Generates a digital signature using the user certificate and task content provided by the Worklist Application

[Figure 28–2](#) provides an example.

Figure 28–2 Digital Signature and Certificate

The following digital signature features are supported:

- PKCS7 signatures based on X.509 certificates
- Browser-based, digitally-signed content without attachments

28.1.10.1 Prerequisites

Prerequisites for using digital signatures and certificates are as follows:

- Users of the Worklist Application 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 following entry in the `workflow-config.xml` file:

```
<workflowConfigurations>
. . .
```

```

. . .
<trustedCAList>
  <trustedCA CAName="CN = Intg, OU =AppServ, O =Oracle, C = US"
    CAURL="http://www.oracle.com/Integration%20CRL%20Data.crl">
  <trustedCA CAName="CN = Intg1, OU =AppServ, O =Oracle, C = US"
    CAURL="http://www.oracleindia.in.com/Integration%20In.crl">
  <trustedCA CAName="CN = Intg2, OU =AppServ, O =Oracle, C = US"
    CAURL="http://www.oracle.us.com/integration.crl">
</trustedCAList>
</workflowConfigurations>

```

28.1.10.2 Interfaces and Methods

Table 28–8 through Table 28–11 describe the methods in the evidence store service. Package `oracle.bpel.services.security.evidence` corresponds to the evidence service.

Table 28–8 *ITaskEvidenceService Interface*

Method	Description
<code>createEvidence</code>	Creates evidence and stores it in the repository for nonrepudiation.
<code>getEvidence</code>	Gets a list of evidences 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.
<code>uploadCertificate</code>	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.
<code>updateEvidence</code>	Updates the CRL verification part of the status. This includes verified time, status, and error messages, if any.
<code>validateEvidenceSignature</code>	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 28–9 *IEvidence Interface*

Method	Description
<code>getCertificate</code>	Gets the certificate used to sign this evidence.
<code>getCreateDate</code>	Gets the creation date of the evidence.
<code>getErrorMessage</code>	Gets the error message associated with the CRL validation.
<code>getEvidenceId</code>	Gets the unique identifier associated with the evidence.
<code>getPlainText</code>	Gets the content that was signed as part of this evidence.
<code>getPolicy</code>	Gets the signature policy of the evidence. This is either <code>PASSWORD</code> or <code>CERTIFICATE</code> .
<code>getSignature</code>	Gets the signature of this evidence.
<code>getSignedDate</code>	Gets the date on which the signature was created.

Table 28–9 (Cont.) IEvidence Interface

Method	Description
getStatus	Gets the CRL validation status. This can be one of the following: <ul style="list-style-type: none"> ■ AVAILABLE — The evidence is available for CRL validation. ■ FAILURE — CRL validation failed. ■ SUCCESS — CRL validation succeeded. ■ UNAVAILABLE — The CRL data could not be fetched. ■ WAIT — CRL validation is in process.
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 task title of the task with which this evidence is associated.
getTaskVersion	Gets the task 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 28–10 ICertificate 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 28–11 PolicyType and WorkflowType Interface

Method	Description
fromValue	Constructs an object from the string representation.
value	Returns the string representation of this object.

See Also:

- [Section 26.6.8.8, "Specifying a Workflow Signature Policy"](#) on page 26-62 for details about specifying digital signatures and digital certificates in the Human Task editor
- [Chapter 31, "Designing Task Display Forms for Human Tasks"](#) for details about digitally signing a task action in the Oracle BPEL Worklist Application

28.2 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: e-mail, telephone voice message, IM, or SMS. Notifications can be sent from a human task in a BPEL process or from a BPEL process directly.

In releases prior to 11g, e-mail notifications were sent through the human task e-mail 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 task e-mail notification layer works with the Oracle Application Server Wireless Notification Service to alert users to changes in the state of a task. The Oracle Application Server Wireless Notification 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

For e-mail notifications, the human task e-mail notification layer works with the Oracle Application Server Wireless Notification Service to enhance message reliability. For the remaining notification channels, you can now use the Oracle Application Server Wireless Notification Service.

On application servers other than Oracle Application Server, the human task e-mail notification layer can be used for e-mail notifications.

This section contains the following topics:

- [Section 28.2.1, "Configuring the Notification Channel"](#)
- [Section 28.2.2, "Contents of Notification"](#)
- [Section 28.2.3, "Configuring Notification Messages in Different Languages"](#)
- [Section 28.2.4, "Sending Actionable Messages"](#)
- [Section 28.2.5, "Error Message Support"](#)
- [Section 28.2.6, "Sending Inbound and Outbound Attachments"](#)
- [Section 28.2.7, "Sending Inbound Comments"](#)
- [Section 28.2.8, "Reliability Support"](#)
- [Section 28.2.9, "Sending Secure Notifications"](#)
- [Section 28.2.10, "Channels Used for Notifications"](#)
- [Section 28.2.11, "Notification Services for SMS, Voice Mail, and IM Channels"](#)
- [Section 28.2.12, "Sending Reminders"](#)

- [Section 28.2.13, "Custom Notification Headers"](#)
- [Section 28.2.14, "Managing the Notification Service"](#)

See Also: The following guides for details about configuring the Oracle Application Server Wireless Notification Service:

- *Oracle Notification Service Administrator's Guide*
- *Oracle Notification Service Developer's Guide*

28.2.1 Configuring the Notification Channel

Configure the notification channel preferences in the following order:

1. Configure the appropriate channel (for example, e-mail) from the messaging server pages of Oracle Enterprise Manager Grid Control Console.
2. Configure the notification service for e-mail and other channels in Oracle JDeveloper.
3. Set the `NotificationMode` parameter for the notification service to either `ALL` or `EMAIL` in the `SOA_ORACLE_`
`HOME\j2ee\home\applications\soa-infra\configuration\workflow-notification-config.xml` file.

Note: For 11g Release 1, this parameter must be configured from the Mailer page in Oracle Enterprise Manager Grid Control Console. Access this page by selecting **Administration > Human Workflow** from the **SOA Infrastructure** menu, then selecting the **Mailer** tab.

By default, this value is set to `NONE`, meaning that no notifications are sent. The possible values for the `NotificationMode` attribute are:

- `ALL` – the e-mail, IM, SMS, and voice channels are configured and notification is sent through any channel.
- `EMAIL` – Only the e-mail channel is configured for sending notification messages.
- `NONE` – No channel is configured for sending notification messages. This is the default setting.

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 as follows:

- **Assigned** — when the task is assigned to users or a group. This action captures the following task actions — *adhoc route, delegate, escalate, information for a task is submitted, push back, reassign, release, and resume.*
- Task is completed
- Task is errored
- Task is expired
- Information is requested for a task
- Task outcome is updated
- Task is suspended

- Task is resumed
- Task is withdrawn
- Task is updated
 - Task payload is updated
 - Task is updated
 - Comments are added
 - Attachments are added and updated
- All Other Actions
 - Any action not covered in any of the actions listed above. This includes acquiring a task.

Notifications can be sent to users involved in the task in various capacities. This includes:

- Assignees – the users or groups to whom the task is currently assigned
- Initiator - the user who created the task
- Creator – the user who created the task
- Approvers – the users who have approved the task so far
 - This applies to a sequential list of approvers participant type where multiple users have approved the task and a notification must be sent to all.
- Owner – the owner of the task

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.

See Also:

- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details about configuring the notification channel
- [Section 26.6.7, "Specifying Participant Notification Preferences"](#) on page 26-53 to configure task notifications in the Human Task editor
- [Chapter 23, "Notifications and the Oracle User Messaging Service"](#)

28.2.2 Contents of Notification

Each e-mail notification can contain the following parts:

- The notification message
- The HTML content from the worklist application — This is a read-only view of the worklist application on the task.
- Task attachments — If the notification includes 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 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 worklist application task details page.

The following example demonstrates the use of `hwf:getNotificationProperty` and `hwf:getTaskResourceBundle` together:

```
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
```

28.2.3 Configuring Notification Messages in Different Languages

A notification consists of four types of data generated from multiples sources and internationalized differently. However, for all internationalizations, the locale is obtained from the `BPMUser` object of the identity service.

- Prepackaged strings (action links, comments, Worklist Application, 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

Use one of the following methods to internationalize messages in the notification content:

- 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 function

```
hwf:getNotificationProperty('locale')
```

The task ID corresponding to a notification can be retrieved with the function

```
hwf:getNotificationProperty('taskId')
```

- If a different resource bundle is used, then use the following XPath extension to retrieve localized messages:

```
orcl:get-localized-string()
```

See Also: [Section 26.6.8.4, "Specifying Multilingual Settings"](#) on page 26-59

28.2.4 Sending Actionable Messages

There several methods for sending actionable messages:

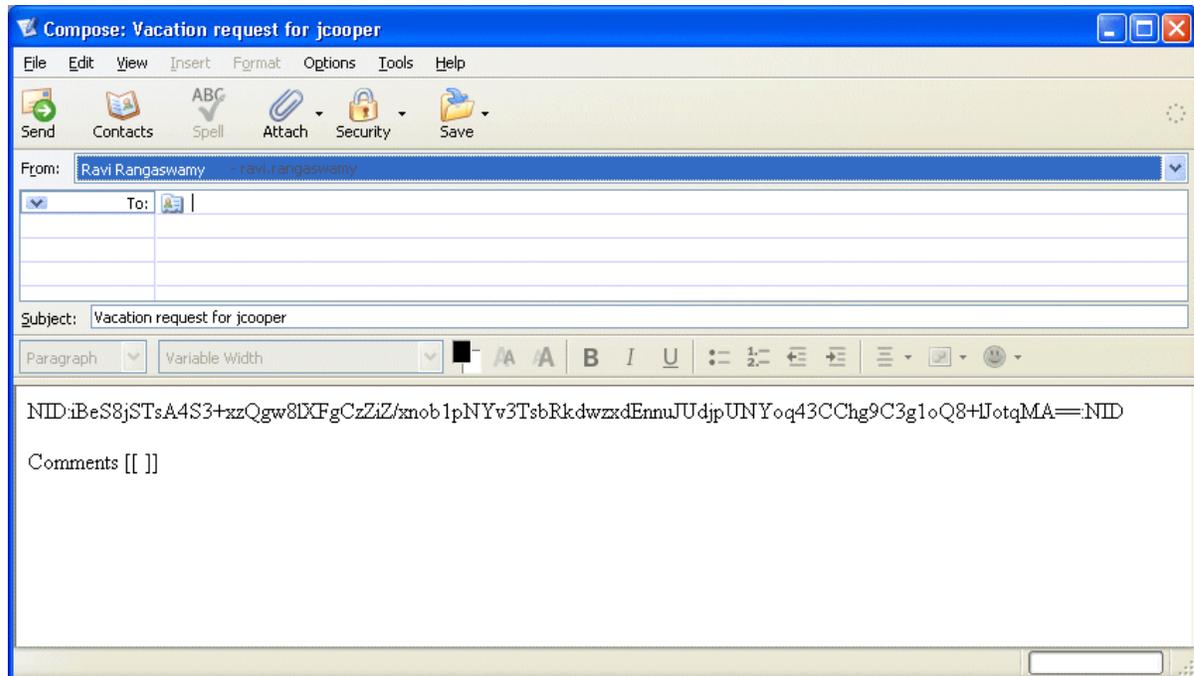
- [Section 28.2.4.1, "Sending Actionable E-mails for Human Tasks"](#)
- [Section 28.2.4.2, "Sending Actionable Instant Messages"](#)

28.2.4.1 Sending Actionable E-mails for Human Tasks

Task actions can be performed through e-mail if the task is set up to enable actionable e-mail (the same actions can also be performed from the Oracle BPEL Worklist Application). An actionable e-mail account is the account in which task action-related e-mails are received and processed. This e-mail account name is identified by the element `actionableEmailAccountName` in the configuration file `SOA_ORACLE_HOME\j2ee\oc4j_soa\applications\soa_infra\configuration\workflow-config.xml`.

1. Select **Make e-mail messages actionable** in the **Notification Settings** section of the Human Task editor to make e-mail notifications actionable. (See [Figure 26–28](#) on page 26-53.) This enables you to perform task actions through e-mail.

If a notification is actionable, the e-mail contains links for each of the custom outcomes. Clicking on the links invokes the compose window of the e-mail client. You do not have to change anything in the subject or the body in this e-mail. If you change the content with the NID substrings, the e-mail is not processed.



2. If you want to send task attachments with the notification message, choose **Select task attachments with email notifications** in the **Notification Settings** section of the Human Task editor.
3. Edit the following properties to configure the e-mail driver in the `SOA_ORACLE_HOME\j2ee\oc4j_`
`soa\application-deployments\nsdriver-email\oc4j-connectors.xml` file. Ensure that you specify the host names of the incoming and outgoing e-mail servers, instead of their IP addresses.

```
<config-property name="ReceiveFolder" value="INBOX" />
<config-property name="OutgoingDefaultFromAddr"
  value="WORKFLOW_MAILER@oracle.com" />
<config-property name="OutgoingPassword" value="" />
<config-property name="OutgoingUsername" value="" />
<config-property name="OutgoingMailServer" value="mail.oracle.com" />
<config-property name="IncomingMailServer"
  value="stmail.oracle.com" />
<config-property name="IncomingUserIDs"
  value="bpuser3_us, bpuser2_us" />
<config-property name="IncomingUserPasswords"
  value="Bpeluser1, Bpeluser1" />
<config-property name="IncomingMailIDs"
  value="bpuser3_us@oracle.com, bpuser2_us@oracle.com" />
<config-property name="OutgoingMailServerPort" value="25" />
<config-property name="OutgoingMailServerTLS" value="false" />
<config-property name="IncomingMailServerPort" value="143" />
<config-property name="IncomingMailServerSSL" value="false" />
```

4. Edit the following properties to configure the human task mailer in the `SOA_ORACLE_HOME\j2ee\oc4j_`
`soa\applications\soa-infra\configuration\workflow-notification-config.xml` file. Setting the `NotificationMode` property to `ALL` or `EMAIL` enables notification.

```
<HWFMailerConfiguration
```

```

xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"
NotificationMode="ALL">
<ASNSConfiguration>
<Name>Default</Name>

<EmailFromAddress>WORKFLOW_MAILER@oracle.com</EmailFromAddress>
<EmailReplyToAddress>bpmuser2_us@oracle.com</EmailReplyToAddress>
<EmailRespondToAddress>bpmuser3_us@oracle.com</EmailRespondToAddress>
<IMRespondToAddress>jabber|bpmuser3_
    us@omdemo4.oraclemobile.com</IMRespondToAddress>

```

5. Set the actionable email account name in the `SOA_ORACLE_HOME\j2ee\oc4j_soa\applications\soa-infra\configuration\workflow-config.xml` file.

```

<actionableEmailAccountName>joe.smith@myaccount.com</actionableEmailAccountName>

```

6. Add the following under the `fmwadmin` user in the `SOA_ORACLE_HOME\j2ee\oc4j_soa\config\system-jazn-data.xml` file.

```

<property name="business_email" value="joe.smith@myaccount.com"/>

```

See Also: [Section 26.6.7.4, "Securing Notifications, Making Messages Actionable, and Sending Attachments"](#) on page 26-56

28.2.4.2 Sending Actionable Instant Messages

Task actions can be performed through IM if the task is set up to enable actionable IM. Task action-related IMs are received and processed in the actionable IM account.

The IM account name is identified by the `IMRespondToAddress` element in the `SOA_ORACLE_HOME\j2ee\home\applications\soa-infra\configuration\workflow-notification-config.xml` configuration file.

Actionable email notification works only when the host name of the mail server is specified.

Ensure that you select **Make e-mail messages actionable** in the **Notification Settings** section of the Human Task editor to make IM notifications actionable. (See [Figure 26-28](#) on page 26-53.)

See Also: [Section 26.6.7.4, "Securing Notifications, Making Messages Actionable, and Sending Attachments"](#) on page 26-56

28.2.5 Error Message Support

The human task e-mail 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 11g Application Server Control Console.

See Also: *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details about viewing messages

28.2.6 Sending Inbound and Outbound Attachments

If the include attachments flag is checked; only e-mail is sent. The e-mails include all the task attachments as e-mail attachments. Select **Send task attachments with e-mail notifications** in the **Notification Settings** section of the Human Task editor. (See [Figure 26–28](#) on page 26-53.)

In the actionable e-mail reply, the user can add attachments in the e-mail and these attachments are added as task attachments.

See Also: [Section 26.6.7.4, "Securing Notifications, Making Messages Actionable, and Sending Attachments"](#) on page 26-56

28.2.7 Sending Inbound Comments

In the actionable e-mail reply, the user can add comments in the e-mail between `Comments [[' and ']]` and those contents are added as task comments. For example, `Comments[[looks good]]`. See Step 1 on page 28-24 for an example.

28.2.8 Reliability Support

The human task e-mail notification layer works with the Oracle Application Server Wireless Notification Service to provide the following reliability support:

- Messages are not lost:
 - If the human task e-mail notification layer crashes after acknowledging receipt of a message from the human task
 - If the human task e-mail notification layer and Oracle Application Server Wireless Notification Service both crash before the Oracle Application Server Wireless Notification Service acknowledges receipt of a message from the human task
 - If the Oracle Application Server Wireless Notification Service is down; message delivery is retried until successful
 - If a notification channel is down
- Once an address has been identified as invalid, it is added to the bad address list. Outgoing notifications are not resent until the address is corrected. The administrator then removes the invalid address from the list.
- 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

See Also:

- [Chapter 23, "Notifications and the Oracle User Messaging Service"](#) for additional details about the reliable notification service
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details about administering notification messages

28.2.9 Sending Secure Notifications

If a notification is marked as secure in the **Notification Settings** section of the Human Task editor, a default notification message is used. (See [Figure 26–28](#) on page 26-53.) 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 the Oracle BPEL Worklist Application. You must log in to see task details.

See Also: [Section 26.6.7.4, "Securing Notifications, Making Messages Actionable, and Sending Attachments"](#) on page 26-56

28.2.10 Channels Used for Notifications

Users can set up preferred notification channels by using the preferences user interface in the worklist application. The channel is dynamically determined by querying the user preference store before sending the notification. If the user preference is not specified, then email channel is used.

See Also: *Oracle Identity Management Guide to Delegated Administration* for more information on the Oracle Delegated Administration Service

28.2.11 Notification Services for SMS, Voice Mail, and IM Channels

Starting with release 11g, you can use Oracle User Messaging Service for notification delivery.

See Also: [Chapter 38, "Configuring User Messaging Service"](#) for details about configuring Oracle User Messaging Service

28.2.12 Sending 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.

You set reminders in the **Notification Settings** section of the Human Task editor. (See [Figure 26–28](#) on page 26-53.) Reminder configuration involves these 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 and EXPIRATION.
- **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.

- The `relativeDate` is `ASSIGNED`, the recurrence is `EVERY`, and the reminder duration is `PT1D`. If 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.

See Also: [Section 26.6.7.3, "Setting Up Reminders"](#) on page 26-55

28.2.13 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.

You create custom notification headers in the **Custom Notification Headers** field of the **Notification Settings** section of the Human Task editor that specify the preferred notification channel to use (such as voice, SMS, pager, and so on). The human task e-mail notification layer provides these header values to the rule-based notification service of the Oracle Application Server Wireless Notification Service for use.

For example, set the **Name** field to `deliveryType` and the **Value** field to `FAX`.

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 IDM. The notification message is created from the information provided by both services.

See Also:

- [Section 26.6.7.4, "Securing Notifications, Making Messages Actionable, and Sending Attachments"](#) on page 26-56 for specifying custom notification headers in the Human Task editor.
- [Chapter 40, "User Messaging Preferences"](#)

28.2.14 Managing the Notification Service

An administrator can perform the following management tasks from Oracle Enterprise Manager 11g Application Server 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 e-mails and block e-mail address for incoming notification responses.

- Manage the bad e-mail address list
- Access runtime data of failed notifications. You can purge this data when it is no longer needed.

See Also: *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details

28.3 Configuring the Assignment Service

This section describes how to configure the assignment service.

This section contains the following topics:

- [Section 28.3.1, "Dynamic Assignment Functions"](#)
- [Section 28.3.2, "Dynamically Assigning Task Participants with the Assignment Service"](#)
- [Section 28.3.3, "Custom Escalation Function"](#)

28.3.1 Dynamic Assignment 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. Three functions are automatically provided. However, you can also create your own functions and register them with the workflow service. [Table 28–12](#) describes the three dynamic assignment functions.

Table 28–12 *Dynamic Assignment Functions*

Function	Description
ROUND_ROBIN	Picks each user or group in turn.
LEAST_BUSY	Picks the user or group with the least number of tasks currently assigned to it.
MOST_PRODUCTIVE	Picks the user or group that has completed the most tasks over a certain time period (by default, the last seven days).

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 (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 task 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 the three functions, a dynamic assignment framework is provided that allows you to implement and configure your own dynamic assignment functions.

28.3.1.1 Implementing a Dynamic Assignment Function

To implement your own dynamic assignment function, 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
```

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 a number of methods that are useful when implementing dynamic assignment functions.

See Also: `SOA_ORACLE_HOME\javadoc\soa-infra` for the Javadoc on dynamic assignment interfaces and utilities

28.3.1.2 Configuring Dynamic Assignment Functions

Dynamic assignment functions are configured along with other human task configuration parameters in the `workflow-config.xml` file in the `SOA_ORACLE_HOME\j2ee\home\applications\soa-infra\configuration` directory. You configure these parameters from the Oracle Enterprise Manager 11g Application Server Control Console.

Each dynamic assignment function must have an entry 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 classpath of the class that implements the function.

In addition, the function tag can optionally contain any number of `<property>` tags. These tags pass initialization parameters to the dynamic assignment function. Each property tag must contain a name attribute. The value of the property is specified in the body of the tag.

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.

See Also: *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details about configuring the dynamic assignment functions from Oracle Enterprise Manager 11g Application Server Control Console

28.3.1.3 Configuring 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 the worklist application to show a list of available dynamic assignment functions.

To specify display names (and appropriate translations) for your dynamic assignment functions, add entries to the resource property file `WorkflowLabels.properties`, and associated resource property files in other languages. This file exists in the `SOA_Oracle_Home\admin\template\bpel\services\wfresource` directory.

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.

28.3.2 Dynamically Assigning Task Participants with the Assignment Service

Human task participants are specified declaratively in a routing slip. The routing slip guides the human task by specifying the participants and how they participate in the human task (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 task.

This section contains the following topics:

- [Section 28.3.2.1, "Assignment Service Overview"](#)
- [Section 28.3.2.2, "Implementing an Assignment Service"](#)
- [Section 28.3.2.3, "Example of Assignment Service Implementation"](#)
- [Section 28.3.2.4, "Deploying a Custom Assignment Service"](#)

28.3.2.1 Assignment Service Overview

The assignment service determines the following task assignment details in a human task:

- 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 the Oracle BPEL Worklist Application, a user can request information for reapproval.
- The users who reapprove the task if reapproval is supported.

The human task 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, `jlonon`.
4. `jstein` approves the task and the assignment service indicates that there are no more users to which to assign the task.

28.3.2.2 Implementing an Assignment Service

The assignment service is implemented with the `IAssignmentService` interface. The human task 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 task. 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 backend 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).

28.3.2.3 Example of Assignment Service Implementation

Notes:

- The assignment service class cannot be stateful because every time human task services need to 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 task service tries to determine the permitted actions for a task.
-
-

You can implement your own assignment service plug-in that the human task service invokes during human task execution.

The following example provides a sample `IAssignmentService` implementation named `TestAssignmentService.java`.

```

/* $Header: TestAssignmentService.java 24-may-2006.18:26:16 rarangas 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)
rarangas 01/30/06 -
*/
/**
 * @version $Header: IAssignmentService.java 29-jun-2004.21:10:35 rarangas Exp
 * $
 * @author rarangas
 * @since release specific (what release of product did this appear in)
 */
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.routingslip.model.*;
import oracle.bpel.services.workflow.metadata.routingslip.model.Participants;
import
oracle.bpel.services.workflow.metadata.routingslip.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;
    }
}

```

}

28.3.2.4 Deploying a Custom Assignment Service

You must 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 the `SOA_ORACLE_HOME\bpel\system\classes` directory and unzip your JAR files in the same directory.
- 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.
-
-

28.3.3 Custom Escalation Function

The custom escalation function enables you to integrate a custom rule in a human task. You create a custom task escalation function and register this with the human task service that uses that function in task definitions. The **Advanced Settings** section of the Human Task editor enables you to integrate the rule in a human task.

See Also: [Section 26.6.8.1, "Specifying Escalation Rules"](#) on page 26-58 for details

28.4 Human Task Service and Identity Service Related XPath Extension Functions

Oracle BPEL Process Manager provides XPath extension functions for use with the human task services and IDM. XPath extension functions mimic XPath 2.0 standards. [Table 28–13](#) lists the supported human task service functions and [Table 28–14](#) lists the supported identity service functions.

Table 28–13 Human Task Service Functions

Function	Description
<code>hwf:clearTaskAssignees()</code>	Clears the task assignees in a task.
<code>hwf:createWordMLDocument()</code>	Creates a Word document by transforming the given XSLT to WordML
<code>hwf:getNotificationProperty()</code>	Gets properties for a particular notification.
<code>hwf:getNumberOfTaskApprovals()</code>	Gets the number of task approvals.
<code>hwf:getPreviousTaskApprover()</code>	Gets the previous task approver.
<code>hwf:getTaskAttachmentByIndex()</code>	Gets the task attachment by attachment index.
<code>hwf:getTaskAttachmentByName()</code>	Gets the task attachment by attachment name.

Table 28–13 (Cont.) Human Task Service Functions

Function	Description
<code>hwf:getTaskAttachmentContents()</code>	Gets the task attachment contents by attachment name.
<code>hwf:getTaskAttachmentsCount()</code>	Gets the number of task attachments.
<code>hwf:getTaskAttachmentByIndex()</code>	Gets the resource string for a particular task
<code>hwf:wfDynamicGroupAssign()</code>	Gets the name of an identity service group, selected according to the specified assignment pattern.
<code>hwf:wfDynamicUserAssign()</code>	Gets the name of an identity service user, selected according to the specified assignment pattern.

Table 28–14 Identity Service Functions

Function	Description
<code>ids:getDefaultRealmName()</code>	Gets the default realm name.
<code>ids:getGroupProperty()</code>	Gets a group property.
<code>ids:getManager()</code>	Gets the manager of a given user.
<code>ids:getReportees()</code>	Gets the direct reportees of the user.
<code>ids:getSupportedRealmNames()</code>	Gets the supported realm names.
<code>ids:getUserProperty()</code>	Gets a user property.
<code>ids:getUserRoles()</code>	Gets the user roles.
<code>ids:getUsersInGroup()</code>	Gets the users in a group.
<code>ids:isUserInRole()</code>	Verifies if a user has a given role.
<code>ids:lookupGroup()</code>	Gets the group object.
<code>ids:lookupUser()</code>	Gets the user object.

28.4.1 Deprecated Human Task Service and Identity Service Functions

[Table 28–15](#) lists the human task and identity service functions that were deprecated for release 10.1.3.

Table 28–15 Deprecated Human Task Service and Identity Service Functions

Human Task Function	Identity Service Functions
<code>ora:getNumberOfTaskApprovals()</code>	<code>ora:getGroupProperty()</code>
<code>ora:getPreviousTaskApprover()</code>	<code>ora:getManager()</code>
<code>ora:getTaskAttachmentByIndex()</code>	<code>ora:getReportees()</code>
<code>ora:getTaskAttachmentByName()</code>	<code>ora:getUserRoles()</code>
<code>ora:getTaskAttachmentContents()</code>	<code>ora:getUsersInGroup()</code>
<code>ora:getTaskAttachmentCount()</code>	<code>ora:isUserInRole()</code>
	<code>ora:lookupGroup()</code>
	<code>ora:lookupUser()</code>
	<code>ora:getUserProperty()</code>

28.5 NLS Configuration

You can specify resource bundles for displaying task details in different languages in Oracle BPEL Worklist Application.

In addition, the resource property file `WorkflowLabels.properties` can be used for setting display names for the following:

- Dynamic assignment functions
- Payload mapping attribute labels
- Task attributes

See Also:

- [Section 26.6.8.4, "Specifying Multilingual Settings"](#) on page 26-59 for details about resource bundles
- [Section 28.1.9.1, "Internationalization of Attribute Labels"](#) on page 28-15 and [Section 28.3.1.3, "Configuring Display Names for Dynamic Assignment Functions"](#) on page 28-32 for additional details about the resource property file `WorkflowLabels.properties`.

28.6 Changes to APIs

Due to licensing issues, use of the `org.exolab.*` packages has been removed. This impacts the following public APIs in the workflow service.

- `oracle.bpel.services.workflow.task.model.Task.getSystemAttributes().getExpirationDuration()` now returns `oracle.bpel.services.workflow.task.impl.Duration` instead of `org.exolab.types.Duration`.
- `oracle.bpel.services.workflow.task.model.Task.getSystemAttributes().setExpirationDuration(...)` now takes `oracle.bpel.services.workflow.task.impl.Duration` as an argument instead of `org.exolab.types.Duration`.
- `oracle.bpel.services.workflow.task.ITaskService.renewTask(...)` now takes `oracle.bpel.services.workflow.task.impl.Duration` as an argument instead of `org.exolab.types.Duration`.

28.7 Summary

This chapter describes how you can integrate systems and services with a human task into a single end-to-end process flow using Oracle BPEL Process Manager. The predefined human task participant types are described, as are the components of human task services—the task service, task routing service, identity service, worklist service, notification service, and others.

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 33.1, "Introduction to Oracle BPM Worklist"](#)
- [Section 33.2, "Logging In to Oracle BPM Worklist"](#)
- [Section 33.3, "Customizing the Task List Page"](#)
- [Section 33.4, "Acting on Tasks: The Task Details Page"](#)
- [Section 33.5, "Setting Vacation and Other Rules"](#)
- [Section 33.6, "Using the Worklist Administration Functions"](#)
- [Section 33.7, "Creating Worklist Reports"](#)
- [Section 33.8, "Accessing Oracle BPM Worklist in Local Languages"](#)
- [Section 33.9, "Summary"](#)

See [Appendix A, "Building a Custom Worklist Client,"](#) for how to use the APIs exposed by the workflow service.

33.1 Introduction to Oracle BPM Worklist

The BPM Worklist application enables 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.

The worklist application 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 the group and route them appropriately.

The worklist is rendered in a browser or in e-mail by a task display form that you create using ADF task flows in Oracle JDeveloper. See [Chapter 31, "Designing Task Display Forms for Human Tasks,"](#) for more information.

33.2 Logging In to Oracle BPM Worklist

Table 33–1 lists the different types of users recognized by Oracle BPM Worklist, based on the privileges assigned to the user.

Table 33–1 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.

See [Section 28.1.5, "Identity Service,"](#) for more information about the predefined roles that can be associated with worklist user types.

33.2.1 How to Log In to the Worklist

Use Internet Explorer 7 or Firefox 2.0.0.2 to access Oracle BPM Worklist.

To log in:

1. Go to

`http://host_name:port_number/integration/worklistapp`

- *host_name* is the name of the host on which Oracle BPEL Process Manager is installed
- The *port_number* used at installation (typically 8888) is noted in

`SOA_Oracle_Home\install\bpelsetupinfo.text`

2. Enter the user name and password.

You can use the preseeded user `fmwadmin` and `welcome1` 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. If more than one realm has been configured, select a realm. (This field does not appear if your organization has only one realm.)
4. Click **Login**.

33.2.2 What Happens When You Log In to the Worklist

Identity service workflow APIs authenticate and authorize logins using a user name, password, and realm set. A realm is defined as an organization—a company, country, division, department, and so on. See [Section 33.6.2, "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 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 33–1](#) shows an example of the worklist for the user jstein (who is a manager in the demo user community).

Figure 33–1 Oracle BPM Worklist—The Task List Page

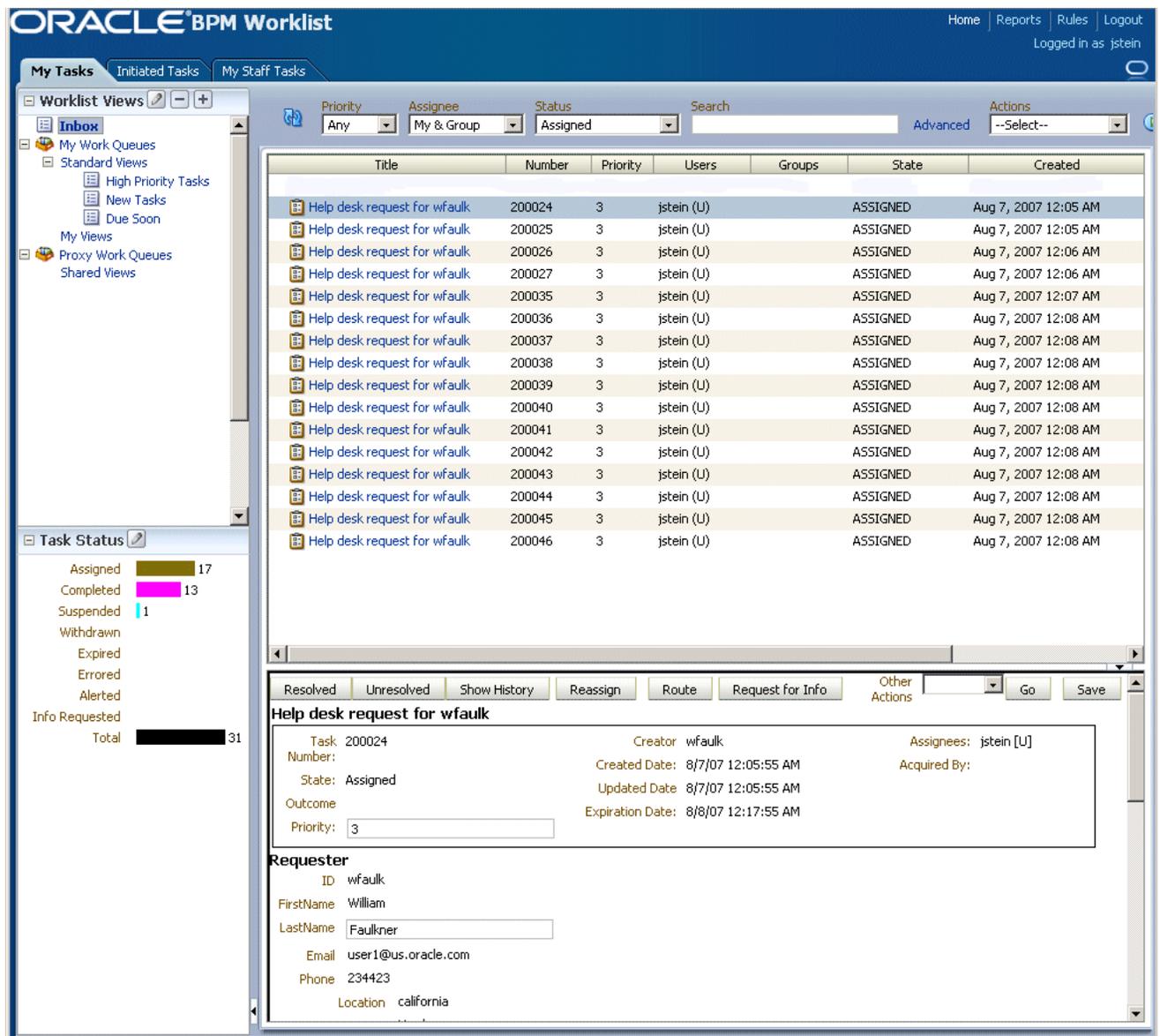


Table 33–2 describes the components of the task list page.

Reviewers—Row 1 of the table is changed. Please review. (12-11-07/db)

Table 33–2 Components of the Task List Page

Component	Description
Tabs	<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 Evident Search tab. <p>See the following for more information:</p> <ul style="list-style-type: none"> ■ Section 33.4.4, "How to Act on Tasks That Require a Digital Signature," for information about evidence search ■ Section 33.6.1, "How to Manage Other Users' or Groups' Rules (as an Administrator)" ■ Chapter 30, "Human Task and Approval Management Integration," for information about approval groups
Links (top right)	<p>Reports—The following reports are available: Unattended Tasks Report, Tasks Priority Report, Tasks Cycle Time Report, Tasks Productivity Report. See Section 33.7.1, "How to Create Reports," for more information.</p> <p>Rules—Set vacation and other rules for users or groups. See Section 33.5.1, "How to Set Vacation and Other Rules," for more information.</p>
Worklist Views	<p>Inbox, My Work Queues, Proxy Work Queues—See Section 33.3.2, "How to Create and Customize Worklist Views," for more information.</p>
Task Status	<p>A bar chart shows the status of tasks in the current view. See Section 33.3.3, "How to Customize the Task Status Chart," for more information.</p>
Display Filters	<p>Specify search criteria from the Priority, Assignee or Status fields. The category filters that are available depend on which tab is selected. From the My Tasks tab, the assignee filters are My, Group, My & Group, and Previous (tasks worked on in the past). From the Initiated Tasks tab, the only assignee filter is Creator. From the My Staff Tasks tab, the only assignee filter is Reportees.</p> <p>Or use Search to enter a keyword. Or use Advanced Search. See Section 33.3.1, "How to Filter Tasks," for more information.</p>

Table 33–2 (Cont.) Components of the Task List Page

Component	Description
Actions List	Select a group action (Claim) or a custom action (for example, Accept and Reject) that was defined for the human task. Other possible actions for a task, such as system actions, are displayed on the task details page for a specific task.
Default Columns	<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.</p> <p>Users—The assignments specified when the human task was created.</p> <p>Groups—The assignments specified when the human task was created.</p> <p>State—One of the following: Assigned, Completed, Errored, Expired, Info Requested, Stale, Suspended, and 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>
Task Details	The lower section of the worklist displays the inline view of the task details page. Buttons indicate available actions. See Section 33.4, "Acting on Tasks: The Task Details Page," for more information.

33.3 Customizing the Task List Page

You can customize your task list in a number of ways, including adding worklist views, selecting which columns to display, and displaying a subset of the tasks based on filter criteria.

33.3.1 How to Filter Tasks

Filters are used to display a subset of tasks, based on the following filter criteria:

- **Priority**—Select from **Any** or **Highest (1)** through **Lowest (5)**.
- **Assignee**—Select from the following:
 - **My**—Retrieves tasks directly assigned to the logged-in user
 - **Group**—Tasks assigned to the groups to which the logged-in user belongs
 - **My & Group**—Tasks assigned to the user and the groups to which the logged-in user belongs
 - **Previous**—Tasks that the logged-in user has updated
- **Status**—Select from the following: **Any**, **Assigned**, **Completed**, **Suspended** (can be resumed later), **Withdrawn**, **Expired**, **Errored** (while processing), **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.

[Figure 33–2](#) shows the filter fields.

Figure 33–2 Filters—Priority, Assignee, Status, Search, and Advanced Search

Task Number	Priority	Assigned Users	Assigned Groups	State	Created Date
200000	3	jstein (U)		Assigned	Apr 17, 2007

The filter criteria for a worklist view are saved when you log out and restored when you log in again.

To filter tasks based on priority, assignee, or status:

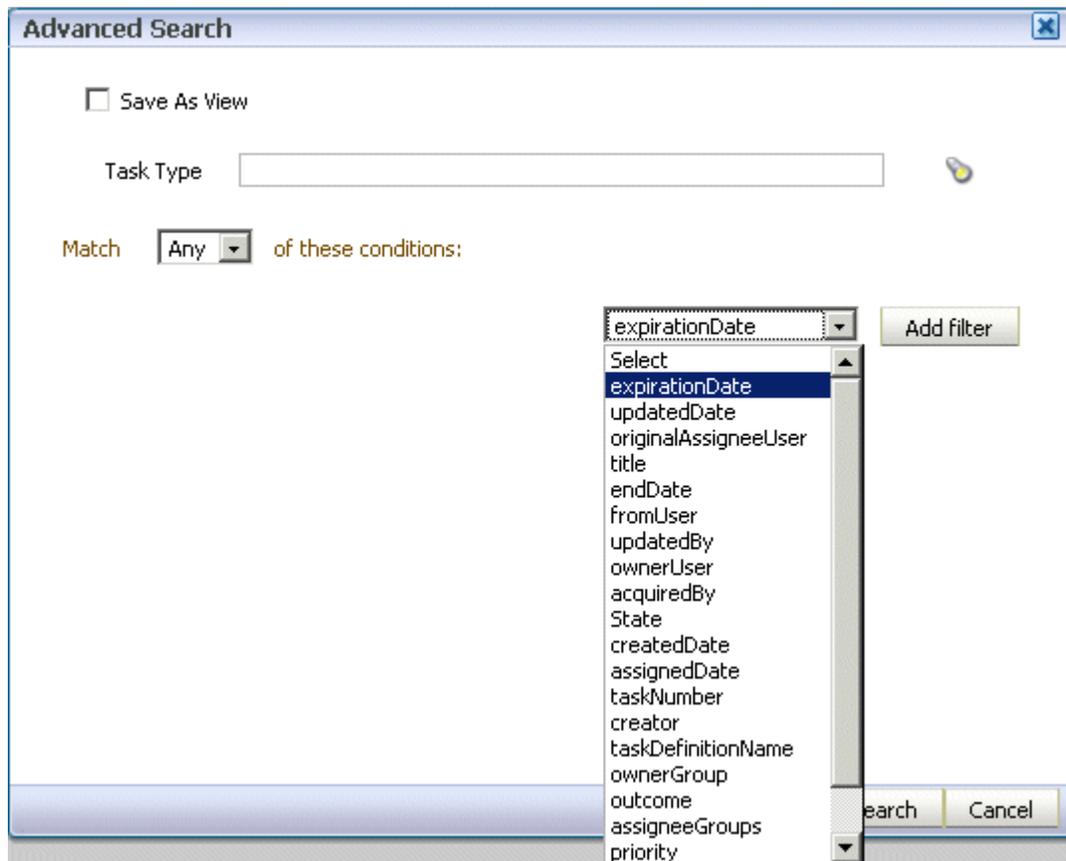
1. Select any combination of options from the **Priority**, **Assignee**, or **Status** lists.
2. Click **Refresh**.

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**.

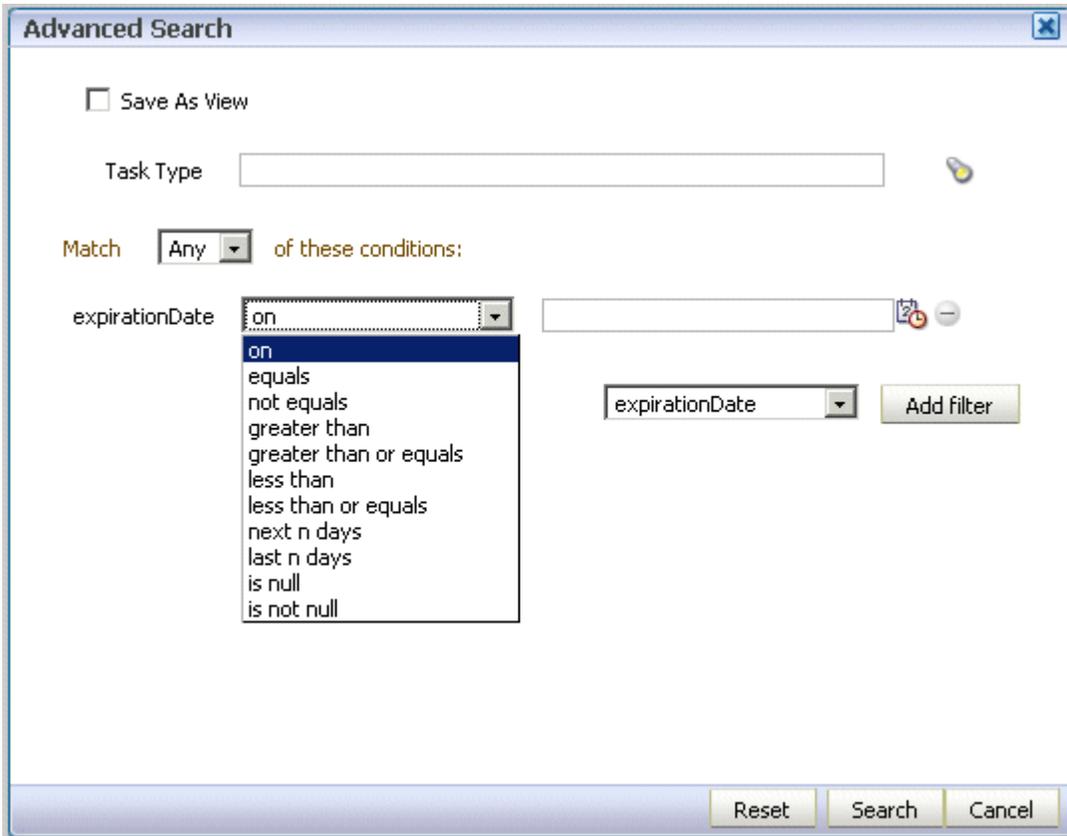
To filter tasks based on an advanced search:

1. Click **Advanced**.
2. Optionally check **Save As View**.
3. Search on a task type.
4. Select **Any** or **All** for matching multiple filters.
5. Add filters, shown in [Figure 33–3](#).

Figure 33–3 Adding Filters for an Advanced Search on Tasks

6. Add parameters, shown in [Figure 33–4](#).

Figure 33–4 Advanced Search



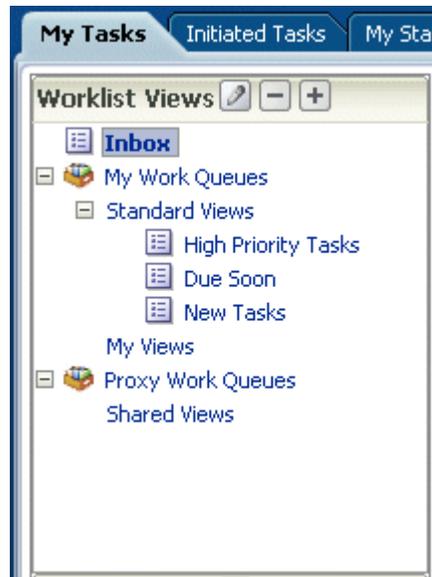
7. Click **Search**.

The task list page with the tasks filtered according to your criteria appears.

33.3.2 How to Create and Customize Worklist Views

The **Worklist Views** area, shown in [Figure 33–5](#), 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 33–5 Worklist Views

Use **Worklist Views** to create, share, and customize views.

Reviewers—The following section was updated. Please review. (12-11-07/db)

To create a worklist view:

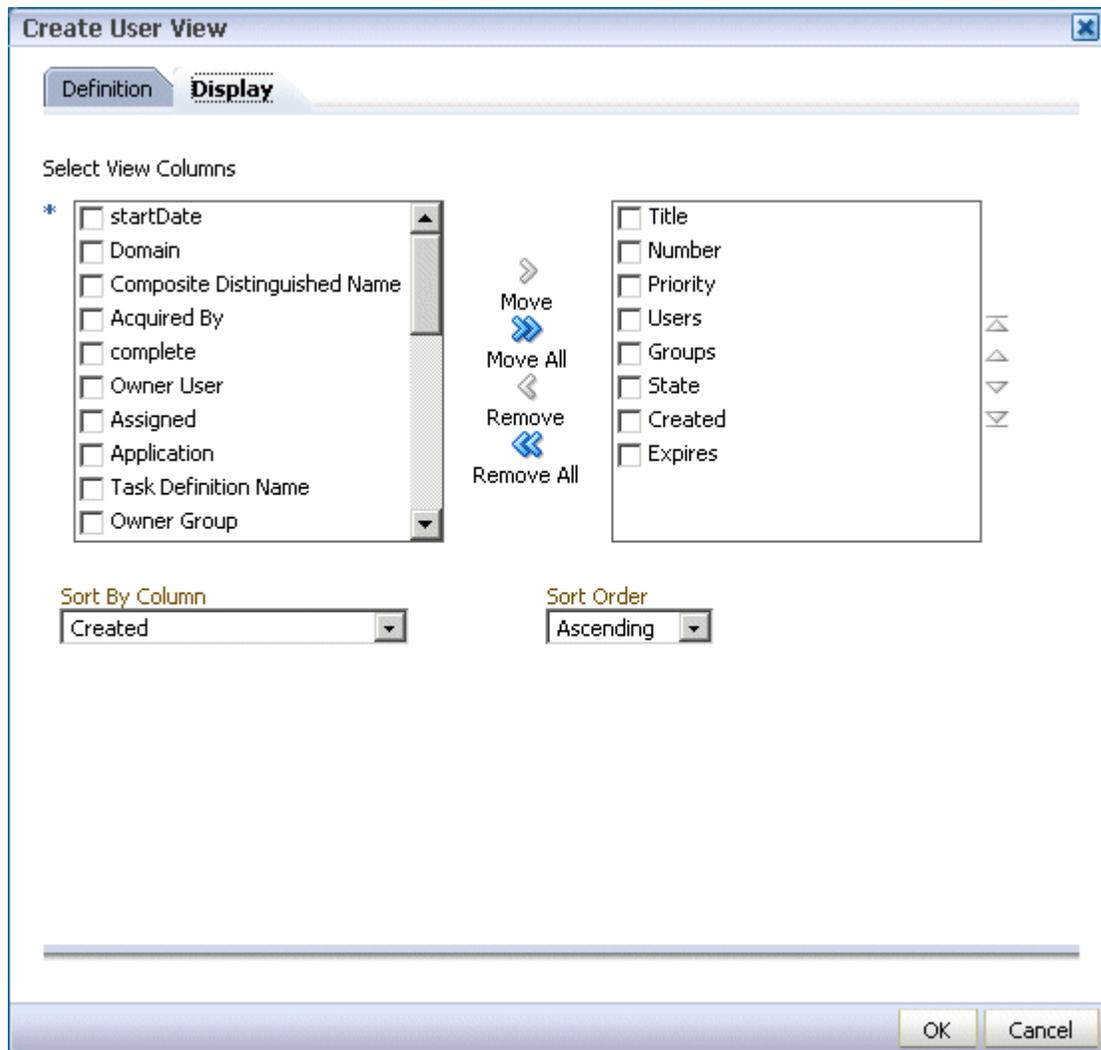
1. In the **Worklist Views** section, click the + icon.
2. Use the **Definition** tab of the Create User View dialog, shown in [Figure 33–6](#).

Figure 33–6 Defining 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.
- **Task Type**—Browse for a task type or leave the field blank for all types.
- **Match any of these conditions**—Select **All** or **Any**.
- **Add Filter**—Select an item from the list and click **Add Filter**. For example, if you select **startDate**, and click **Add Filter**, then a calendar and a list including **on**, **equals**, **not equals**, **greater than**, **less than**, and so on appears.
- **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 33–6](#), to customize the fields that appear in the view.

Figure 33–7 *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**.

Reviewers—The following section was updated. Please review. (12-11-07/db)

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 33–8](#) and [Figure 33–9](#), and click **OK**.

Figure 33–8 Customizing a Worklist View

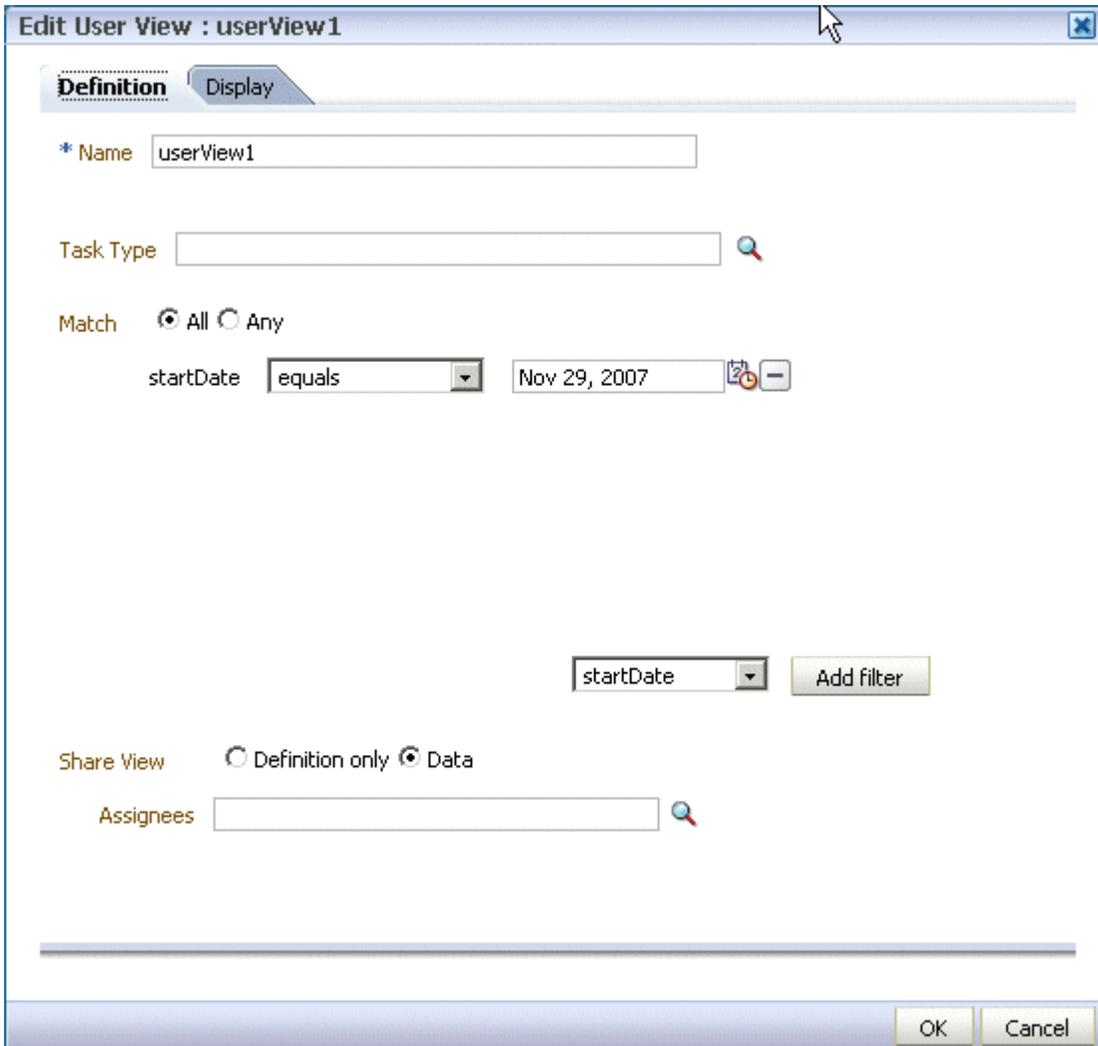
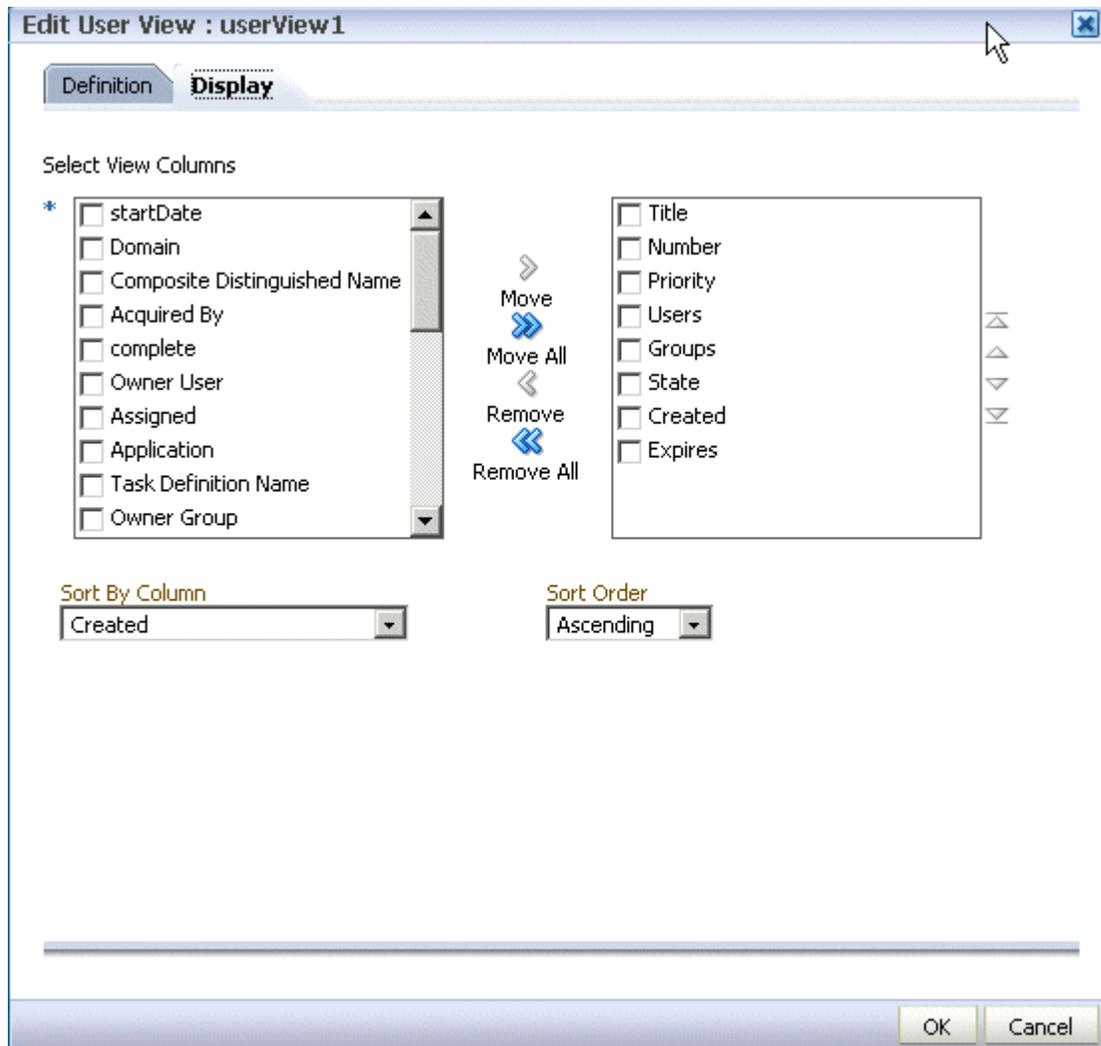


Figure 33–9 Customizing Fields in a Worklist View



Reviewers—What happened to the Number of tasks per fetch field? Can users set the number of tasks that appear in the task display list? (12-11-07/db)

33.3.3 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 33–10](#), and click **OK**.

Figure 33–10 Customizing the Task Status Chart

33.3.4 How to Create a To-Do Task

Use the Create To-Do Task dialog, shown in [Figure 33–11](#), to create a to-do list for yourself or others. List items can be ad hoc tasks, created irrespective of a task definition, or they can be subtasks to break down a business task into measurable subtasks, or they can be associated with a task definition. To-do tasks associated with a task definition inherit task default values such as priority and category if those values are not specified for the to-do task. To-do tasks appear in the assignee's **Inbox**.

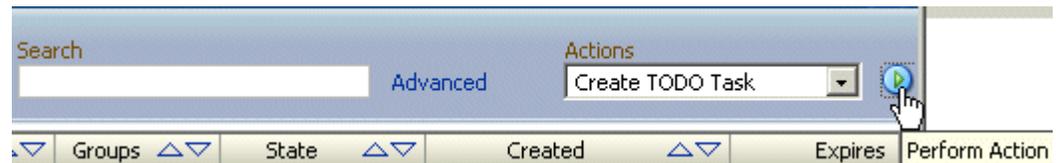
You can create to-do tasks that are children of other to-do tasks or business tasks. When all child to-do tasks are 100% complete, the parent to-do task is also marked as completed. If the parent is a business task, it is not marked as completed. You must set the outcome and complete it.

To-do tasks can be reassigned, escalated, and so on, as well as deleted (logical delete) and purged (physical delete).

Figure 33–11 The Create To-Do Task Dialog

To create a to-do list:

1. From the **Actions** list, select **Create To-Do Task** and click **Perform Action**, as shown in [Figure 33–12](#).

Figure 33–12 Creating a To-Do List

2. Provide details in the Create To-Do Task dialog, shown in [Figure 33–13](#), and click **OK**.
 - **Task Title** and **Category**: Enter anything that is meaningful to you.
 - **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.
 - **Assignee**: You can assign yourself or someone else.

Figure 33–13 Creating a To-Do Task

33.4 Acting on Tasks: The Task Details Page

Task details can be viewed inline (see the lower section in [Figure 33–1, "Oracle BPM Worklist—The Task List Page"](#)) or in a pop-up browser window. (Click the task title *link* to open the pop-up window; click any other column or the task icon to view the inline details page.)

The task details page, shown in [Figure 33–14](#), has the following components:

- Action bar—Displays buttons for custom actions that are defined in the human task, such as setting task outcomes (for example, resolving a help desk request or approving a loan request). The **Other Actions** list displays other system actions—possible actions that do not require additional input that users can take depending on their privileges. For example, actions such as **Renew**, **Suspend**, and **Escalate** often appear under **Other Actions**. For the task initiator or a manager, **Withdraw** may also appear.
- Header—Displays task attributes, including the task title, number, state, priority, assignees, and other flex fields. It also displays dates related to task creation, last modification, and expiration, and information about who created, updated, claimed, or is assigned to the task.
- Payload—Displays the task form that was generated using Oracle ADF. The fields displayed reflect how the human task was created; for example, a loan application in the Loan Demo sample or support ticket details in the Help Desk Request sample.
- Comments—Displays comments entered by various users who have participated in the workflow. A newly added comment and the comment writer’s username 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.
- History—Displays the approval sequence and the update history for the task. See [Section 33.4.2, "Task History,"](#) for more information about the full history and the graphical view, available from the **Show History** button.

Figure 33–14 Task Details Page

Resolved Unresolved Show History Reassign Route Other Actions Go »

Help desk request wfault

Task Number: 200001	Creator	Assignees: jstein [U]
State: Assigned	Created Date: 4/12/07 4:42:44 PM	Acquired By:
Outcome	Updated Date: 4/12/07 4:42:44 PM	
Priority: <input type="text" value="3"/>	Expiration Date: 4/13/07 4:52:44 PM	

ID wfault
 FirstName William
 LastName Faulkner
 Email user1@us.oracle.com
 Phone

Location California
 Type Hardware
 ProblemDescription Unable to reboot the system
 Severity
 Status
 Comment
 ResolvedBy

<div style="border: 1px solid #ccc; padding: 5px;"> Comments Add </div>	<div style="border: 1px solid #ccc; padding: 5px;"> Attachments Add </div>
---	--

Short History

Version	Action	State:	Outcome
1	TASK_VERSION_REASON_INITIATED	ASSIGNED	

A user can view a task when associated with the task as one of the following: current assignee (directly or by group membership), current assignee’s manager, creator, 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 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.

33.4.1 System Actions

The action bar displays system actions, which are available on all tasks based on the user's privileges. [Table 33-3](#) lists system actions.

Table 33-3 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.
Pushback	Use this action to send a task up 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, <code>oracle.tip.worklist.samples.taskactin.renew.duration</code> , in the file <code>pc.properties</code> , which appears in <code>SOA_Oracle_Home\bpel\system\services\config</code> . The default value is P7D (seven days).

Table 33–3 (Cont.) System Task Actions

Action	Description
Submit More Information and Request More Information	Use these actions if another user requests that you supply more information or if you want 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 at present, 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. Buttons that update a task are disabled after suspension.
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.

33.4.2 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 in

```
SOA_Oracle_Home\bpel\system\services\config\wf_config.xml
```

- Acquire
- Adhoc 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 full history is available from the **Show History** button. The full history provides a graphical view of a task flow, as shown in [Figure 33–15](#).

Figure 33–15 Full History: Graphical View

Full History :



The full history also provides the details of each action on a task, as shown in [Figure 33–16](#).

Figure 33–16 Full History: Details

History Details						
Sequence	Participant	Action	State	Action Date		
1	wfaulk	TASK_VERSION_REASON_INITIATED	ASSIGNED	Aug 6, 2007 1:25 AM		
2	jstein	TASK_VERSION_REASON_INITIATED	ASSIGNED	Aug 6, 2007 2:09 AM		
3	jstein	TASK_VERSION_REASON_INITIATED	ASSIGNED	Aug 6, 2007 2:13 AM		
4	jstein	TASK_VERSION_REASON_INITIATED	ASSIGNED	Aug 6, 2007 3:20 AM		
5	jstein	TASK_VERSION_REASON_OUTCOME_UPD	OUTCOME_UPD	Aug 6, 2007 3:25 AM		
6	jstein	TASK_VERSION_REASON_COMPLETED	COMPLETED	Aug 6, 2007 3:25 AM		

The full history provides a snapshot of the task details, as shown in [Figure 33–17](#).

Figure 33–17 Full History: Snapshot of Task Details

Snapshot of Task Details : Help desk request for wfaulk

Help desk request for wfaulk

Task Number: 200000	Creator: wfaulk	Assignees: jstein [U]
State: Assigned	Created Date: 8/6/07 1:25:42 AM	Acquired By:
Outcome	Updated Date: 8/6/07 2:13:35 AM	
Priority: <input type="text" value="3"/>	Expiration Date: 8/7/07 1:37:41 AM	

Requester

ID: wfaulk
 FirstName: William
 LastName:
 Email: user1@us.oracle.com
 Phone: 234423
 Location: california

33.4.3 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 choose 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.

To reassign a task:

1. Click **Reassign**.
2. Provide an individual user or group name, as shown in [Figure 33–18](#).

Figure 33–18 Reassigning a Task

ORACLE BPM Worklist
Logged in as jstein

Reassign Task

Users

Available	Selected
<input checked="" type="checkbox"/> All	<input checked="" type="checkbox"/> All
	<input checked="" type="checkbox"/> wfaulk

Details

Name: William Faulkner	Title: Vice President
Work 123456789	Manager: cdickens
Phone:	Reportees: jstein
Cell 123456789	achrist
Phone:	sfitzger

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.

3. Move the names to the **Selected** area and click **OK**.

To request information:

1. Click **Request for Info**.
2. Provide an individual user or group name, or push back the task to the previous assignee, as shown in [Figure 33–19](#).

Figure 33–19 Requesting Information from a User or Group

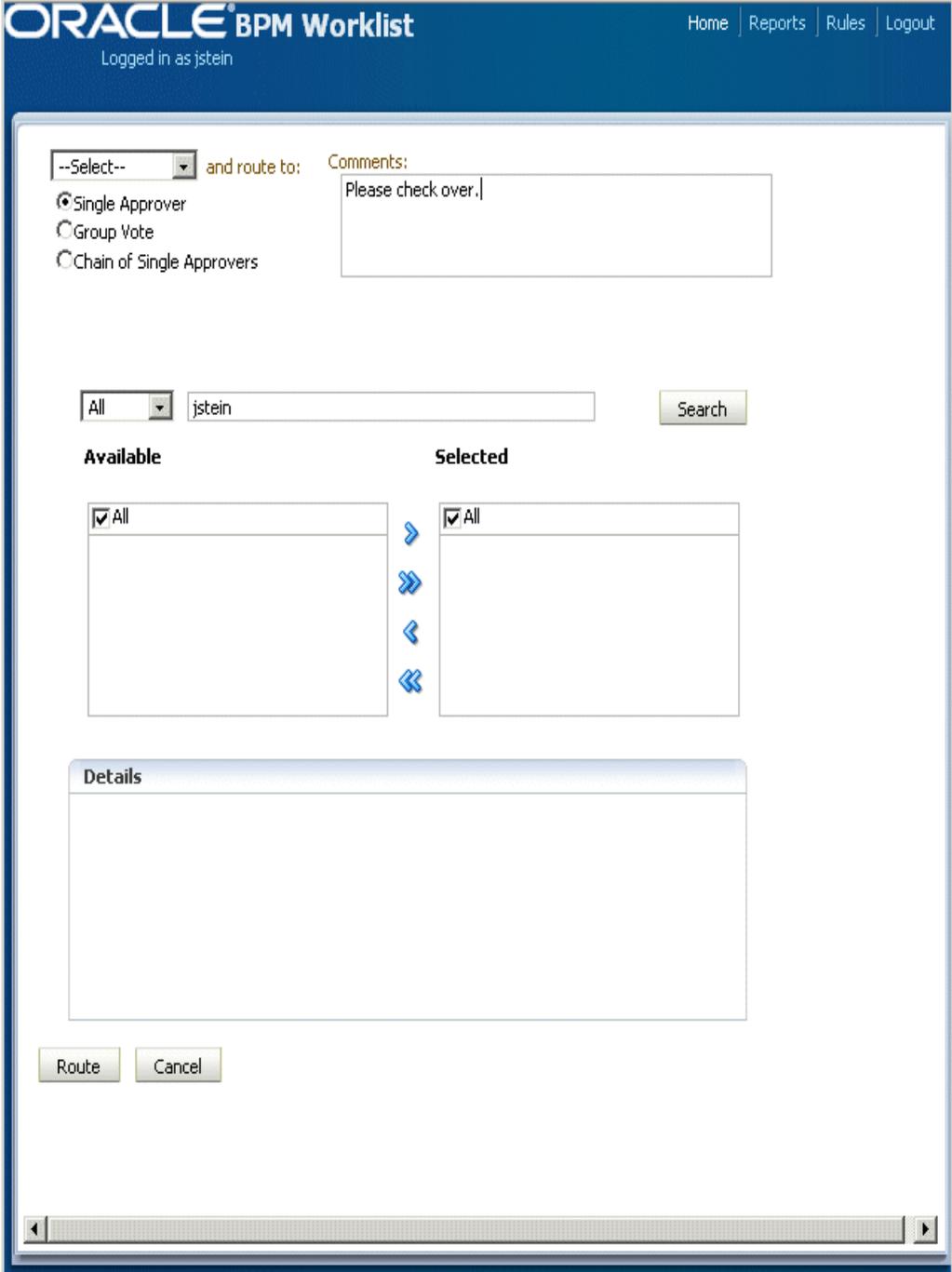
The screenshot shows the Oracle BPM Worklist interface. The header includes the Oracle logo and 'BPM Worklist', with navigation links for Home, Reports, Rules, and Logout. The user is logged in as 'wfaulk'. The main content area has two radio button options: 'Request information from [jstein] and retrace approval chain' (which is selected) and 'Push task back to previous assignee and then to me'. Below these is a 'Comments' field containing the text 'Please provide hardcopy receipt.' and a 'Reapproval Needed' checkbox. At the bottom left is a 'Request Info' button.

3. Click **Request Info**.

To route a task:

1. Click **Route**.
2. Select an action and provide a routing option, as shown in [Figure 33–20](#).

Figure 33-20 Routing a Task



- **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 take action 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.

- **Chain of Single Approvers:** Use this option for a sequential list of approvers. The list can comprise any users or groups. (Users are not required to be part of an organization hierarchy.)
3. Provide user or group names. move the names to the **Selected** area, and click **OK**.
 4. For a group vote, provide consensus information, as shown in [Figure 33–21](#).

Figure 33–21 Providing Consensus Information

Default outcome

Consensus Percentage

Minimum agreement to override Default

5. Click **Route**.

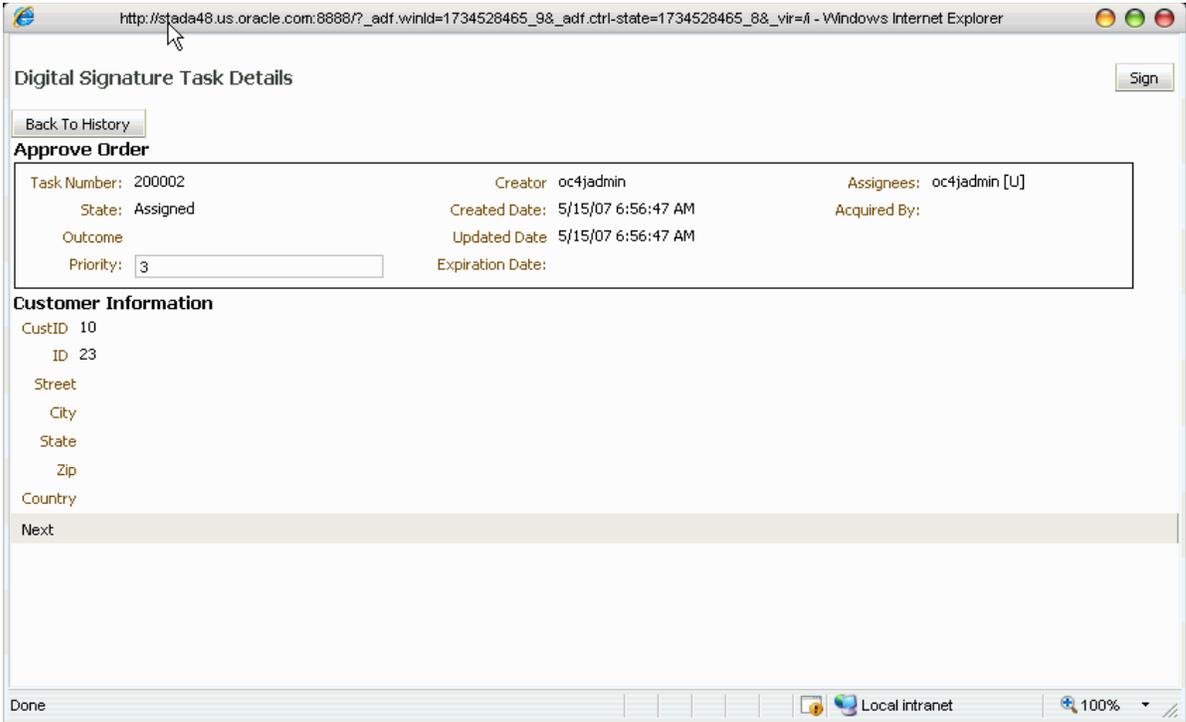
33.4.4 How to Act on Tasks That Require a Digital Signature

The worklist supports the signature policy created in the human task:

- **No signature required** — Participants can send and act upon 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 upon 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.

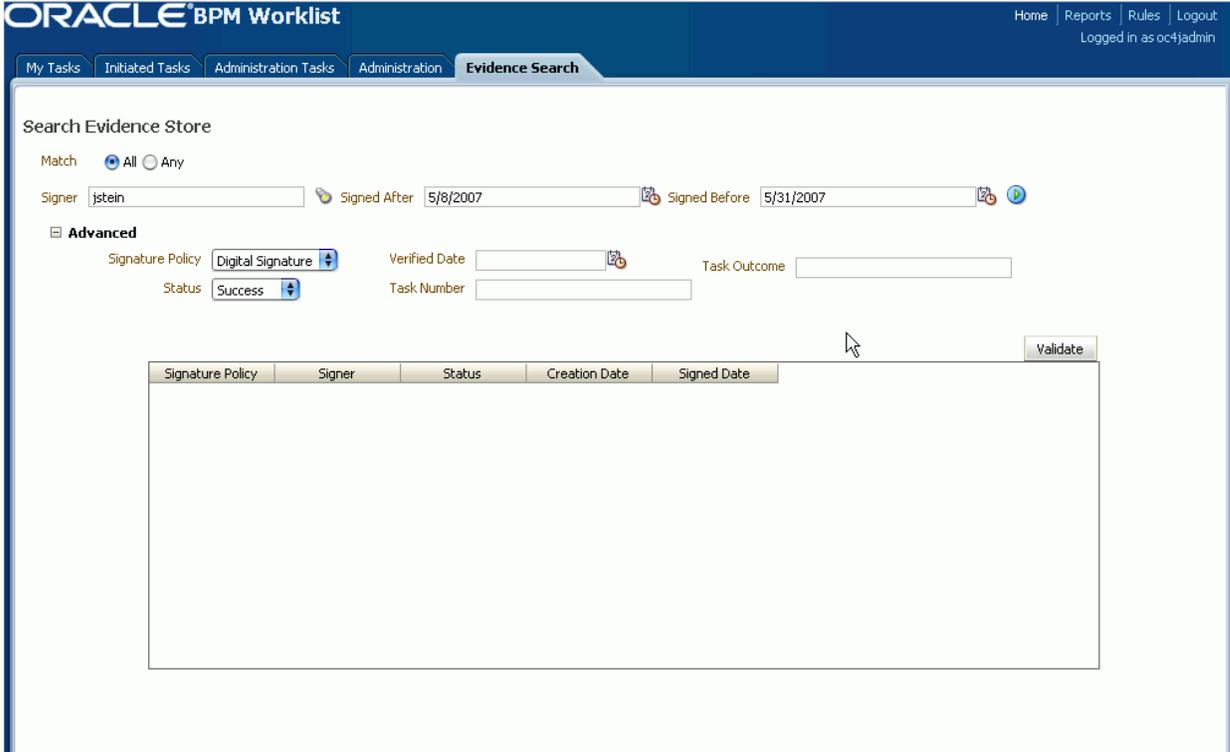
When you act on a task that has a signature policy, the **Sign** button appears, as shown in [Figure 33–22](#).

Figure 33–22 Digital Signature Task Details



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 33–23.

Figure 33–23 The Evidence Store

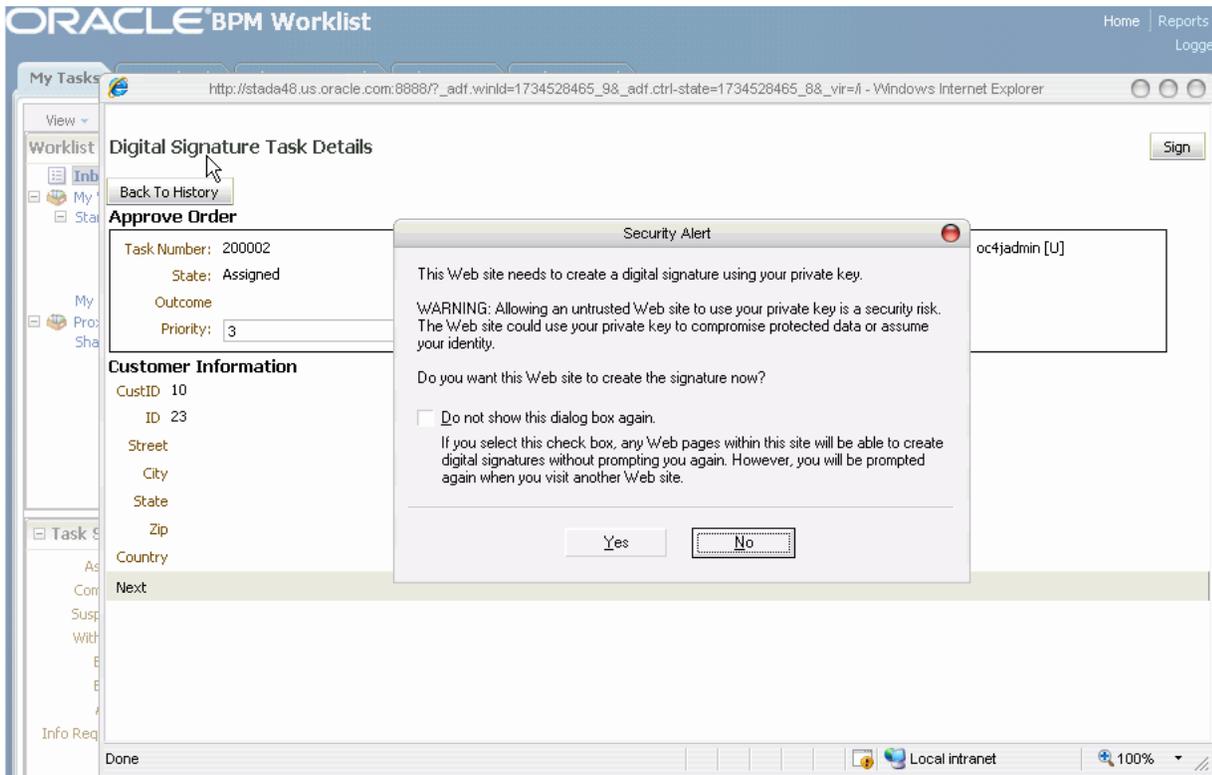


See [Section 28.1.10, "Digital Signatures and the Evidence Store Service,"](#) for more information.

To provide a digital signature:

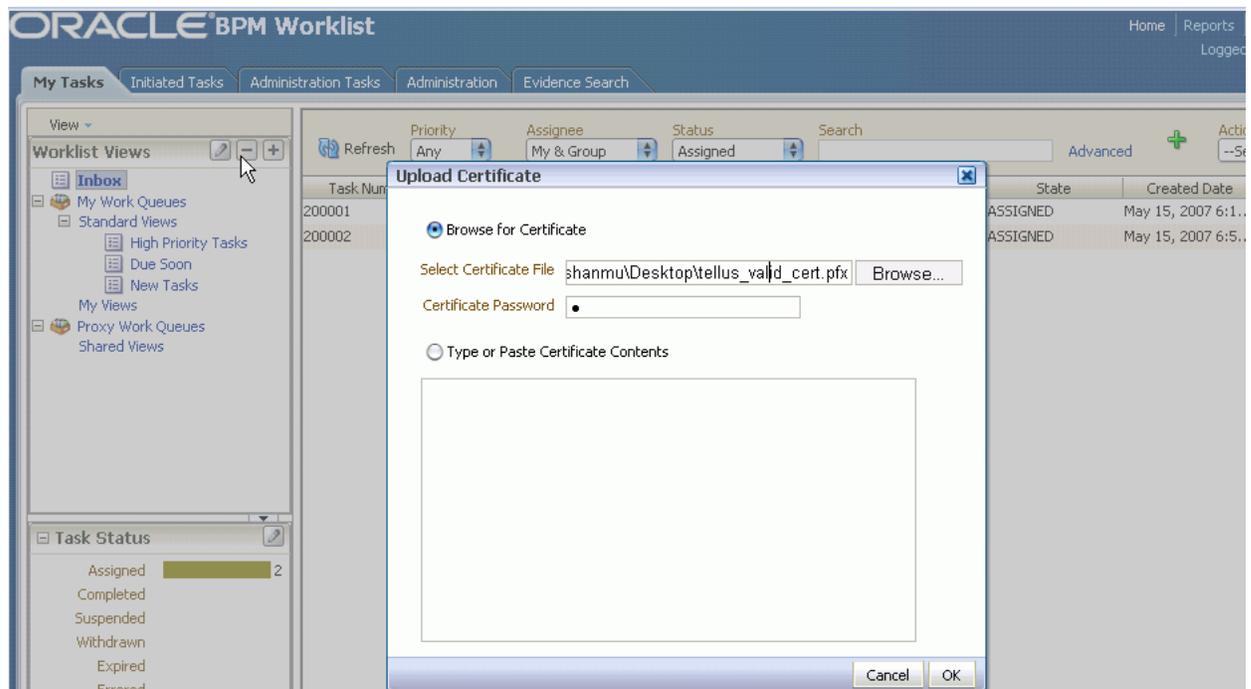
1. For a task that has a signature policy, click **Sign**.
2. To authorize the creation of a digital signature, click **Yes**, as shown in [Figure 33–24](#).

Figure 33–24 Providing a Digital Signature



3. Upload the certificate you need, as shown in [Figure 33–25](#).

Figure 33–25 Uploading a Certificate



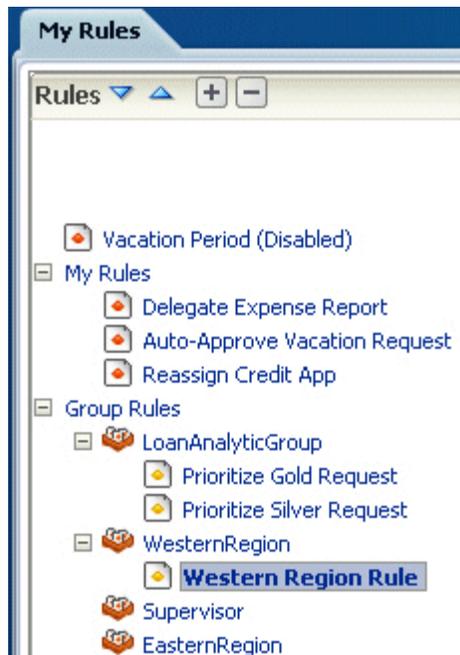
33.5 Setting Vacation and Other Rules

Rules act on tasks, either a specific task type or all the tasks assigned to a user or group.

A rule cannot always apply in all circumstances in which it is used. For example, if a rule applies to more than one task type, 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 33–26](#).

Figure 33–26 Rules



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.

33.5.1 How to Set Vacation and Other Rules

You specify the following when creating a rule:

- Rule name
- Which task or task type the rule applies to—If unspecified, then the rule applies to all tasks.
- 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 as well as any flex fields that have been mapped for the specific tasks. See [Section 33.6.3, "How to Map Flex Fields,"](#) for more information.

User rules do one of 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, if you want 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 will prevent reassignment for loan requests.

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

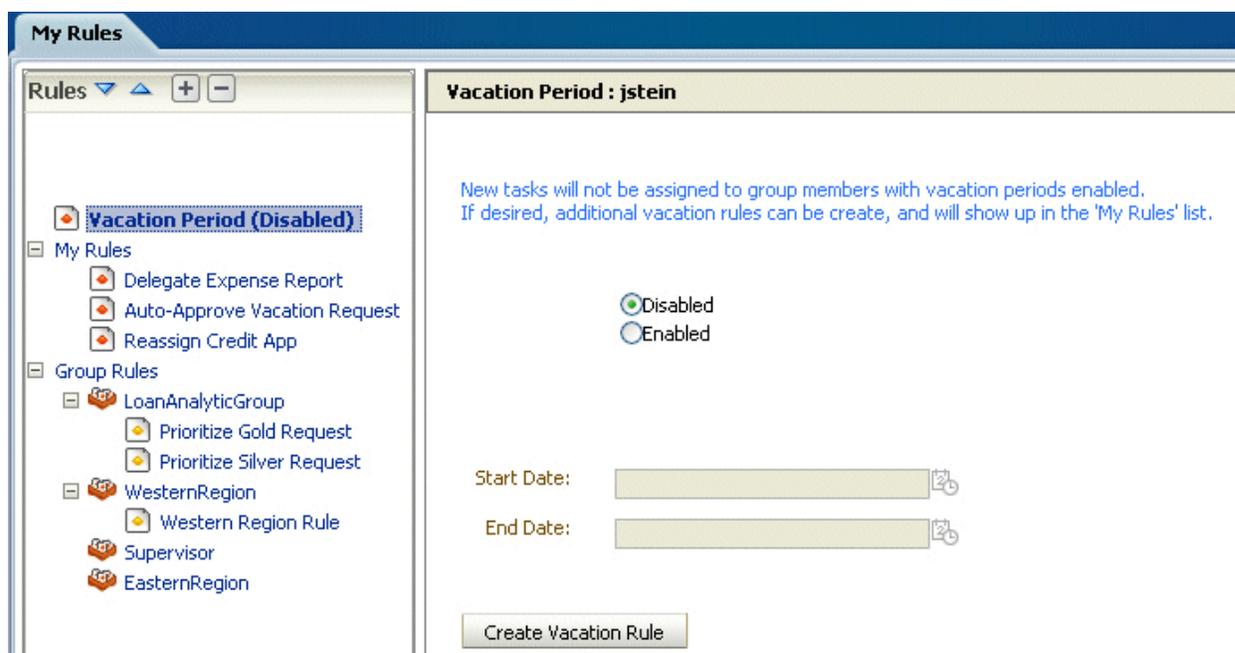
Group rules do one of the following actions:

- **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 vacation rule:

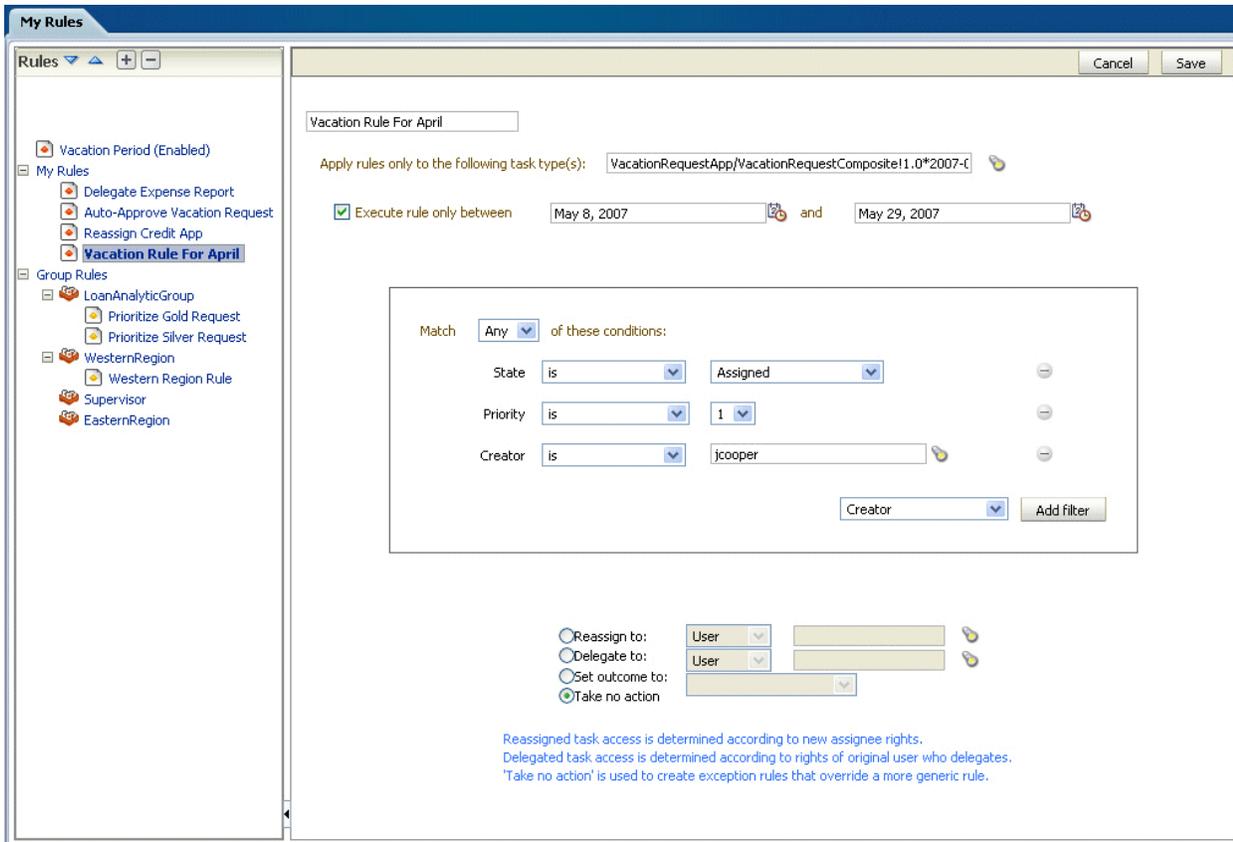
1. On the task list page, click **Rules** (upper right area).
2. Click **Vacation Period (Disabled)**, as shown in [Figure 33–27](#)

Figure 33–27 Creating a Vacation Rule



3. Click **Enabled** and provide start and end dates.
4. Click **Create Vacation Rule**.
5. Provide a name for the rule.
6. Browse for task types to which the rule applies.
7. Provide rule execution dates.
8. Select the actions to be taken (or none) while on vacation (**Reassign to**, **Delegate to**, and so on), as shown in [Figure 33–28](#).

Figure 33–28 *Selecting Vacation Actions*



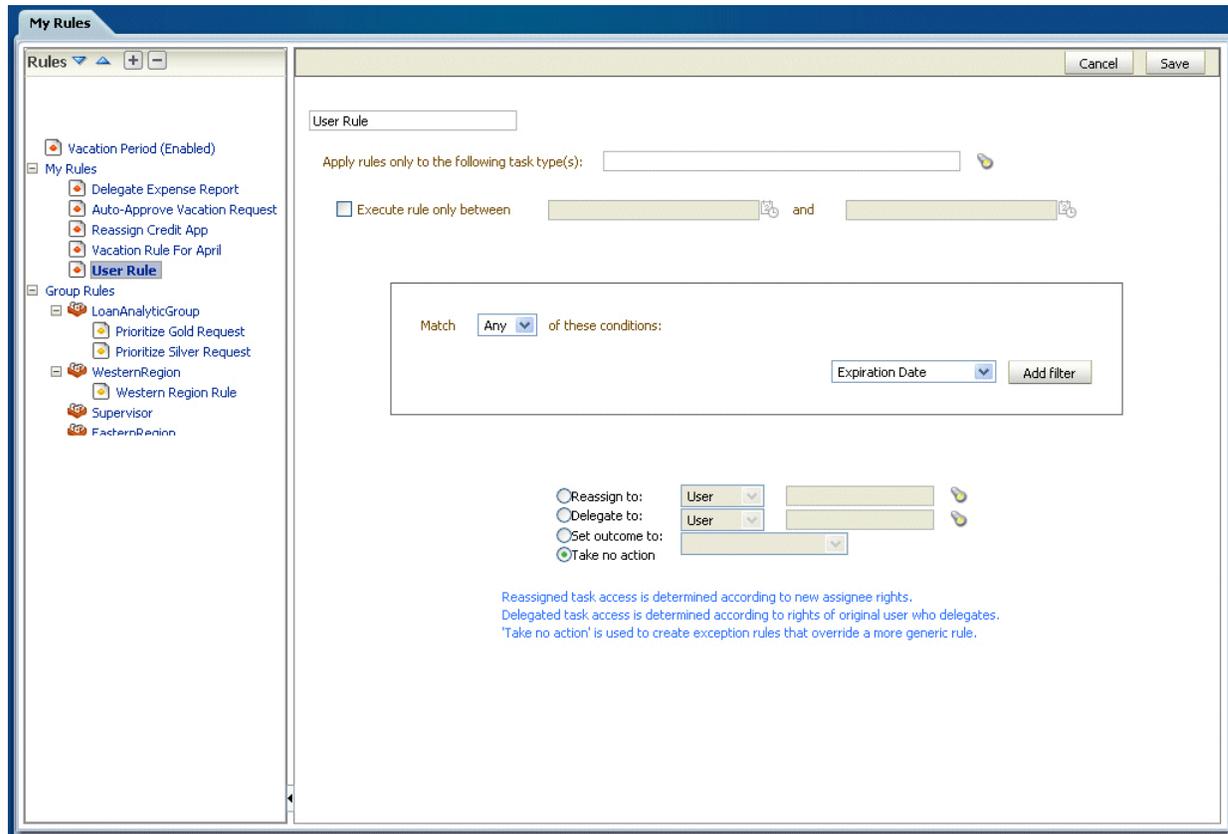
9. Click **Save**.
The new vacation rule appears under the **My Rules** node.

To create a user rule:

1. Click the **Add** icon.
2. Provide rule information and click **Save**.
 - Provide a name for the rule.
 - Browse for task types to which the rule applies.
 - Provide rule execution dates.
 - Set rule conditions.

- Select the actions to be taken (or none) (**Reassign to**, **Delegate to**, **Set outcome to**, or **Take no action**), as shown in [Figure 33–29](#).

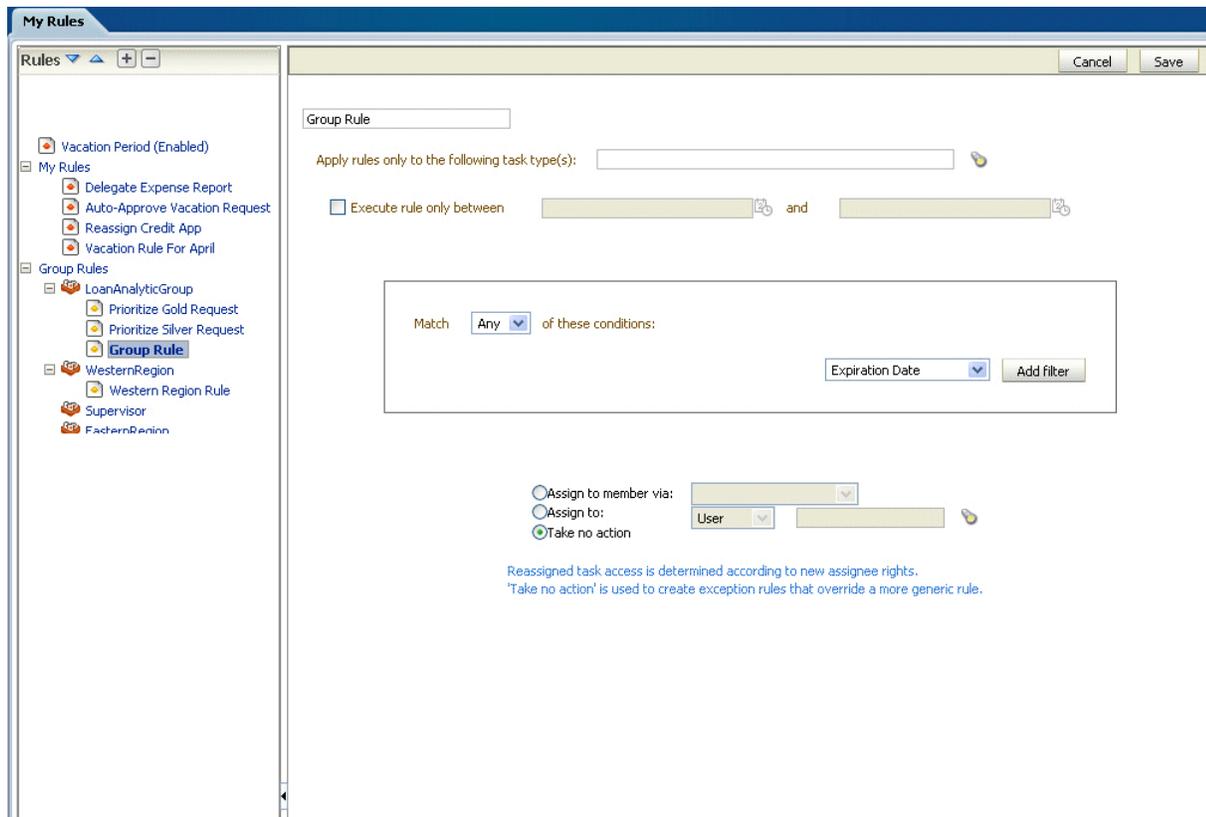
Figure 33–29 *Selecting User Rule Actions*



The new rule appears under the **My Rules** node.

To create a group rule:

1. Click a group name under **Group Rules**.
2. Click the **Add** icon.
3. 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.
 - Set rule conditions.
 - Select the actions to be taken (or none) (**Assign to member via**, **Assign to**, or **Take no action**), as shown in [Figure 33–30](#).

Figure 33–30 Selecting Group Rule Actions

The new rule appears under the **Group Rules** node.

33.6 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.

33.6.1 How to Manage Other Users' or Groups' Rules (as an Administrator)

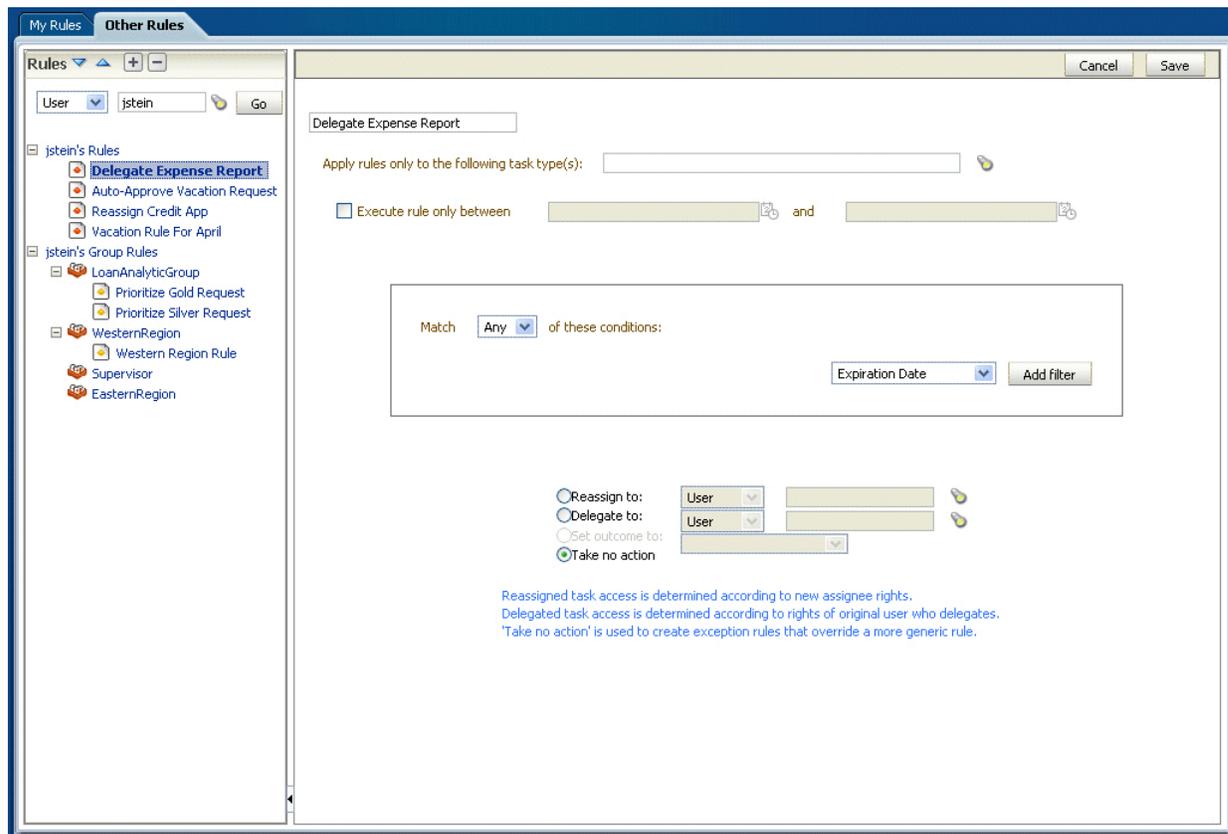
This function is useful for fixing a problem with a rule. Also, for a user who no longer works for the company, administrators can set up a rule for that user so that all tasks assigned to the user are automatically assigned to another user or group.

To create a rule for another user or group:

1. From the task list page, click the **Rules** link (upper right area).
2. Click the **Other Rules** tab.
3. Search for the user or group for whom rules are to be created, as shown in [Figure 33–31](#).

Figure 33–31 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 33–32](#), and click **Save**.

Figure 33–32 Defining Rules for Another User or Group

See [Section 33.5.1, "How to Set Vacation and Other Rules,"](#) for details about the fields on the Other Rules dialog.

33.6.2 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
```

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).

The screenshot shows the Oracle BPM Worklist Administration console. The top navigation bar includes tabs for 'My Tasks', 'Initiated Tasks', 'Administration Tasks', and 'Administration'. The 'Administration' tab is selected. On the left, a tree view shows 'Administration' expanded to 'Application Preferences'. The main content area is titled 'Application Preferences' and contains three configuration fields:

Parameter	Value
Login page realm label	LABEL_LOGIN_REALM
Global branding icon	/images/branding.gif
Resource bundle	oracle.bpel.worklistapp.resource.WorklistResourceBundle

Buttons for 'Cancel' and 'Save' are visible in the top right corner of the configuration area.

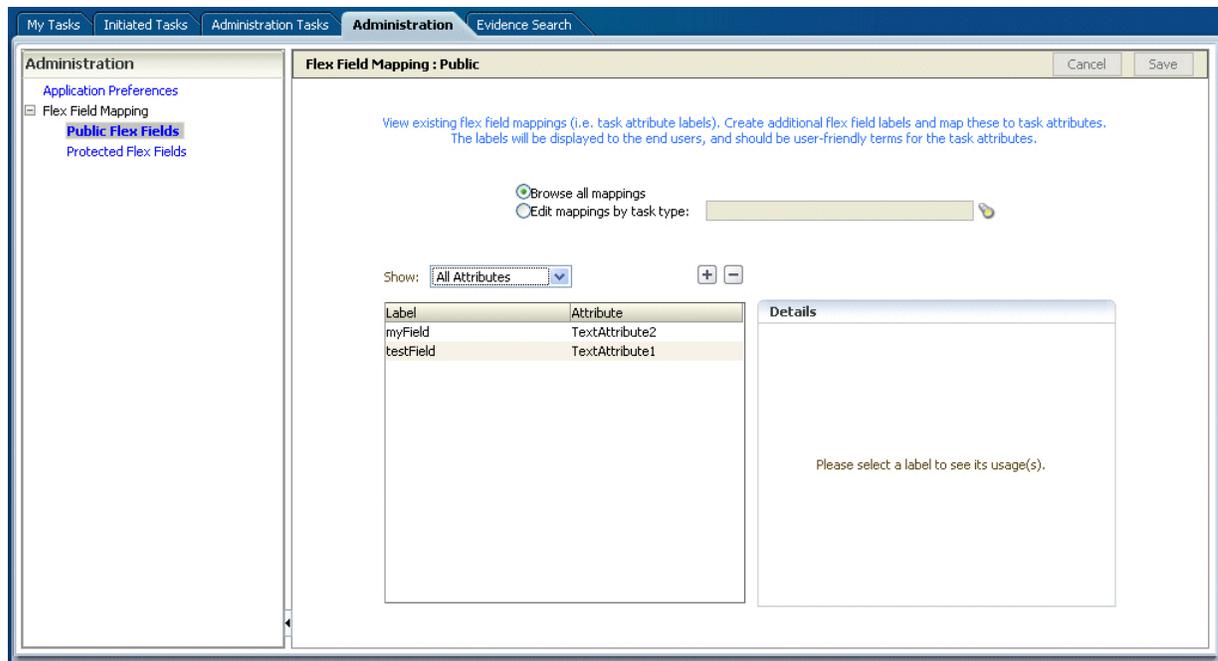
4. Click **Save**.

33.6.3 How to Map Flex Fields

An administrator, or users with special privileges, use flex field mapping, shown in [Figure 33–33](#), 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.

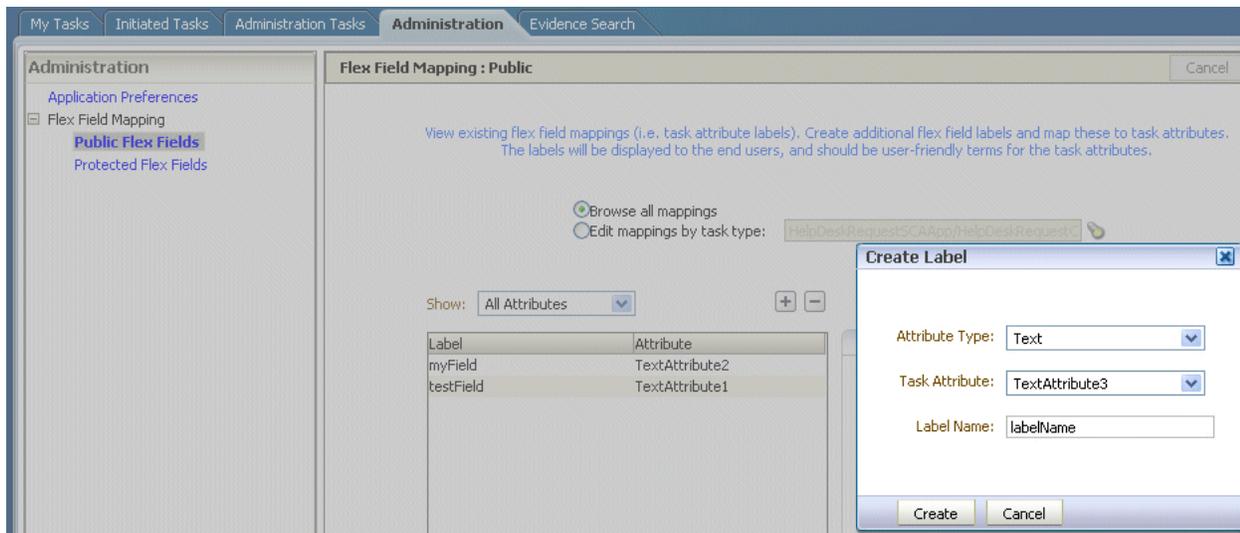
Administrators can map both public flex fields and protected flex fields. Users who have been granted the `workflow.mapping.publicFlexField` privilege can map public flex fields, and see a **Public Flex Fields** node under the Administration tab. Users who have been granted the `workflow.mapping.protectedFlexField` privilege can map protected flex fields, and see a **Protected Flex Fields** node under the Administration tab.

Figure 33–33 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 the **Add** icon (+) to use the Create Label dialog, shown in [Figure 33–34](#).

Figure 33–34 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 end 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.

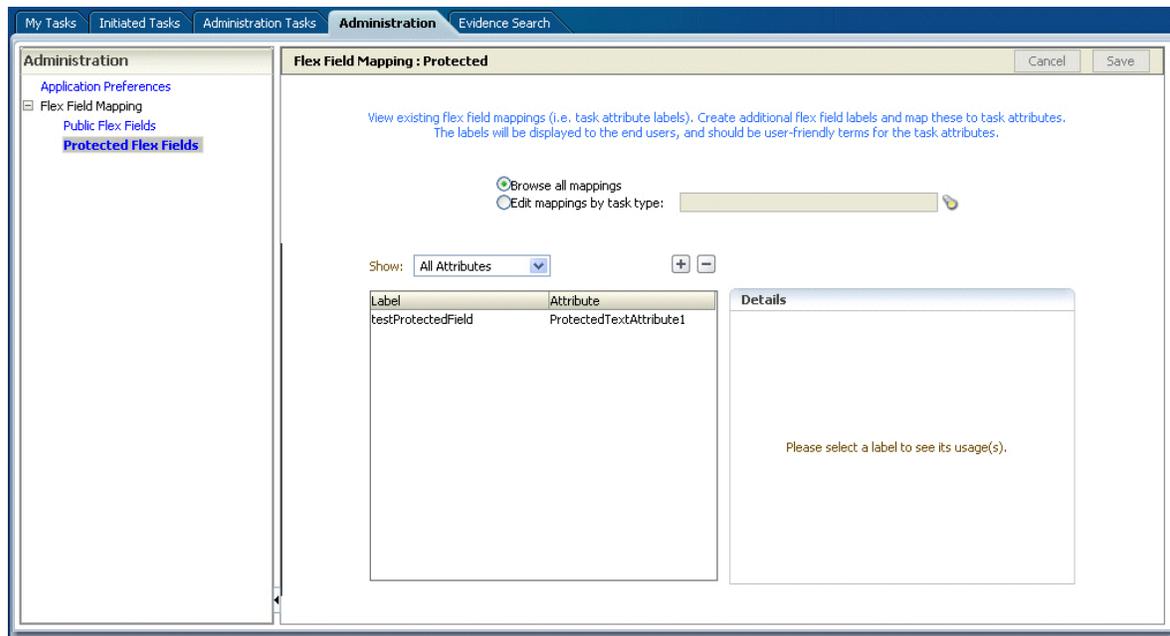
Note the following restrictions:

- Only simple type payload attributes can be mapped. Mapping specific simple types within a complex type is not supported.
- 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:

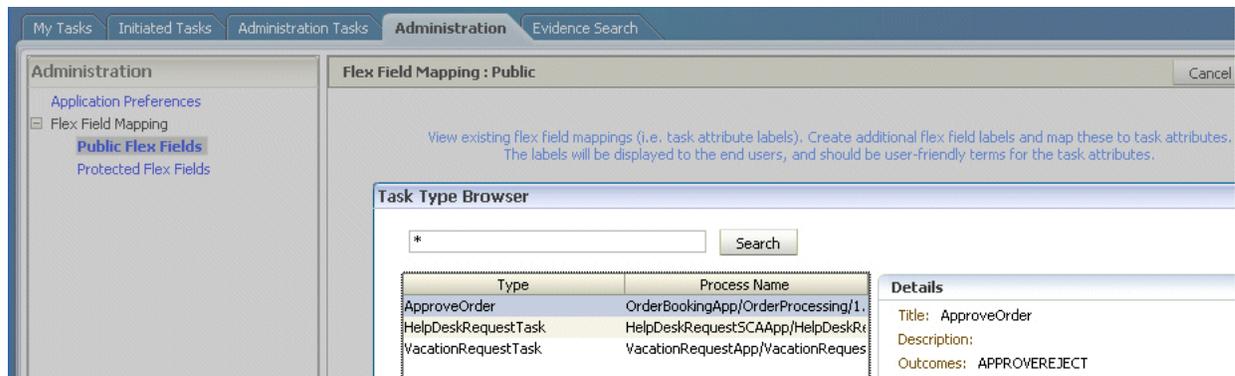
1. Click **Browse all mappings**.
2. Select a row in the label table to display all the payload attributes mapped to a particular label.

Figure 33–35 Browsing Mappings



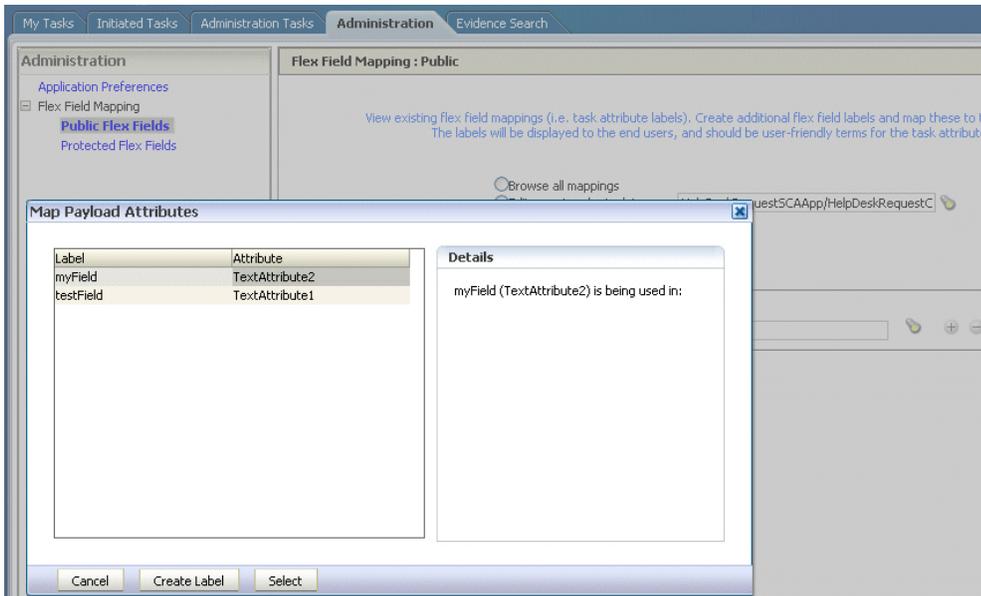
To edit mappings by task type:

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 33–36](#).

Figure 33–36 Selecting a Label



4. Select a mapping label and click **Select**.

Figure 33–37 shows a completed mapping.

Figure 33–37 Flex Field Mapping Created



See [Section 28.1.9.1, "Internationalization of Attribute Labels,"](#) for more information.

See Also: ["Internationalization of Attribute Labels"](#)

33.7 Creating Worklist Reports

The reports listed in [Table 33–4](#) are available for task analysis.

Reviewers—The last row of the table is new. Please review. (12-11-07/db)

Table 33–4 Worklist Report Types

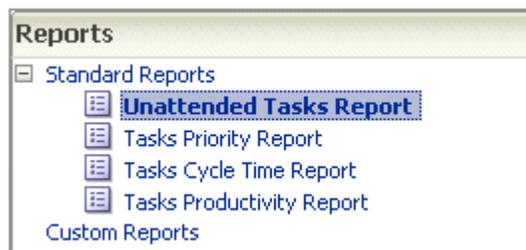
Report Name	Description	Input Parameters
Unattended Tasks	Provides an analysis of tasks assigned to users' groups or reportees' groups that have not yet been claimed (unattended tasks).	<ul style="list-style-type: none"> ▪ Assignee—This option (required) selects tasks assigned to the user's group (My 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 Info_Requested.
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 choose, 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 ▪ Task Priority—The priority (optional) can be Any, 1, 2, 3, 4, or 5.
Tasks Cycle Time	Provides an analysis of the time taken to complete tasks from creation to completion based on users' groups or reportees' groups.	<ul style="list-style-type: none"> ▪ Assignee—Depending on the assignee that you choose, 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 ▪ Task Priority—The priority (optional) can be Any, 1, 2, 3, 4, or 5.
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 chooses, this required option includes the user's tasks (My & Group) or tasks assigned to groups to which the user's reportees belong (Reportees). ▪ 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 chooses, this required option includes the user's tasks (My & Group) or tasks assigned to groups to which the user's reportees belong (Reportees). ▪ 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).

33.7.1 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.



3. Provide inputs to define the search parameters of the report.

Unattended Tasks Report

Provides an analysis of tasks assigned to user's or reportee's groups that need attention, since they have not yet been acquired

Assignee

Creation Date

Expiration Date

Task State

Task Priority

See [Table 33-4](#) for information about input parameters.

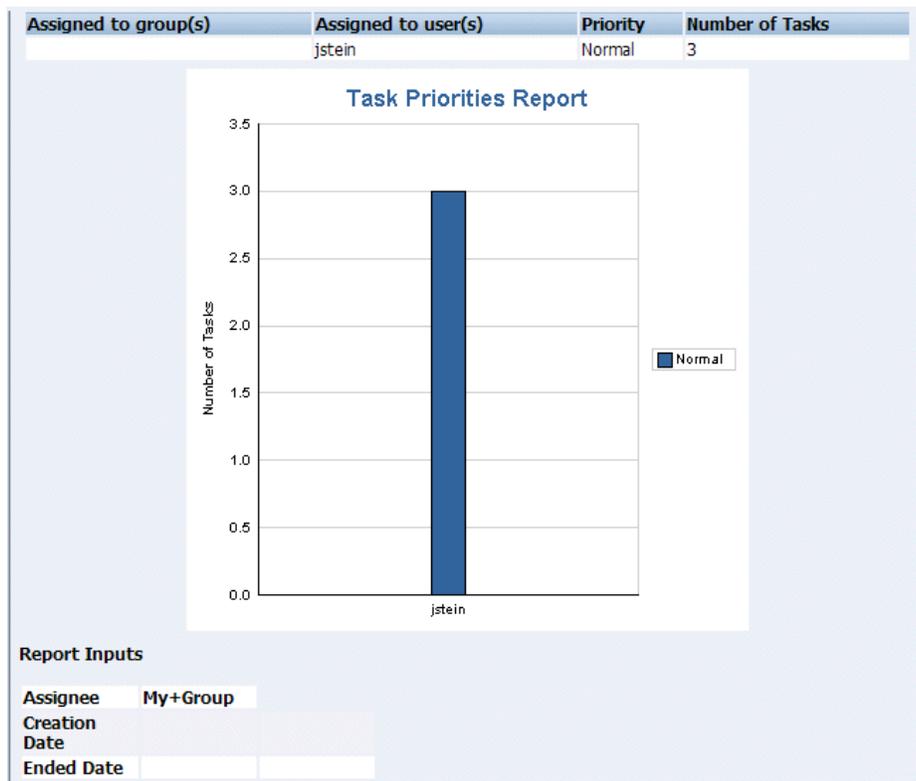
4. Click **Run**.

33.7.2 What Happens When You Create Reports

As shown in [Figure 33-38](#), 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).

Reviewers--These report output images are from 10.1.3. Have they changed for 11g? If so, how can I get new images? (12-11-07/db)

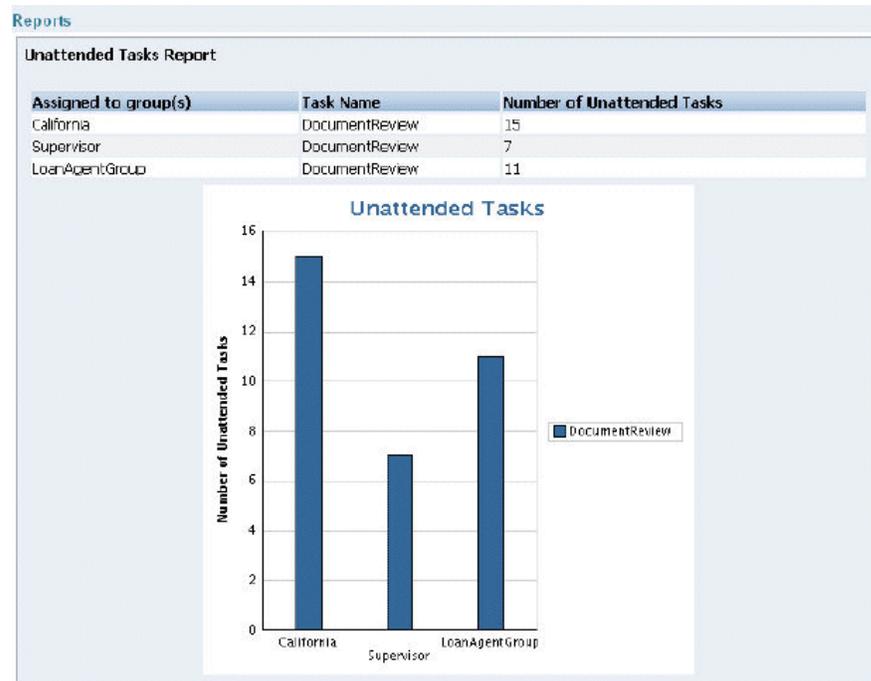
Figure 33-38 Report Display—Table Format, Bar Chart Format, and Report Inputs



33.7.2.1 Unattended Tasks Report

Figure 33–39 shows an example of an Unattended Tasks report.

Figure 33–39 Unattended Tasks Report

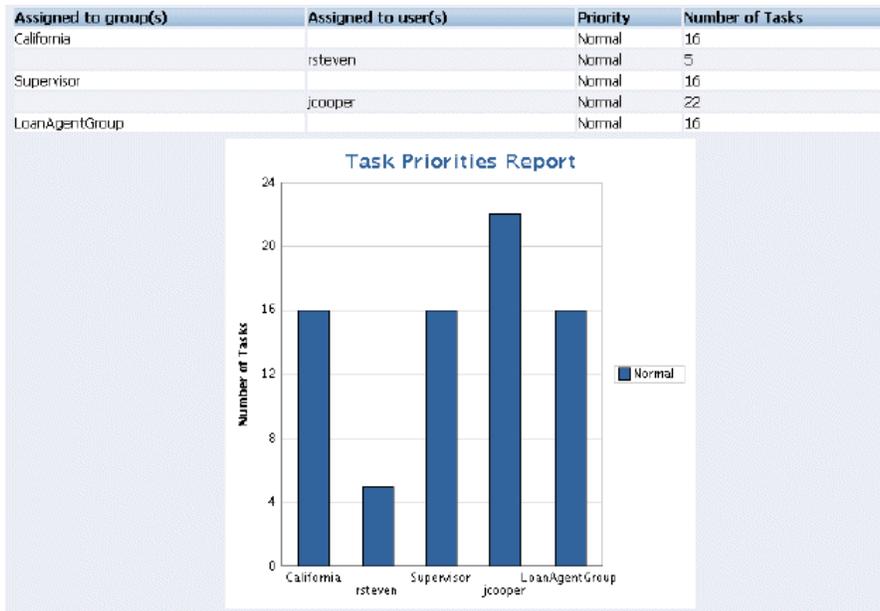


The report shows that the California group has 15 unattended tasks, the Supervisor group has 7 unattended tasks, and the LoanAgentGroup has 11 unattended tasks. The unattended (unclaimed) tasks in this report are all DocumentReview tasks. If more than one type of unattended task exists when a report is run, all task types are included in the report, and the various task types are differentiated by color.

33.7.2.2 Tasks Priority Report

Figure 33–40 shows an example of a Tasks Priority report.

Figure 33–40 Tasks Priority Report

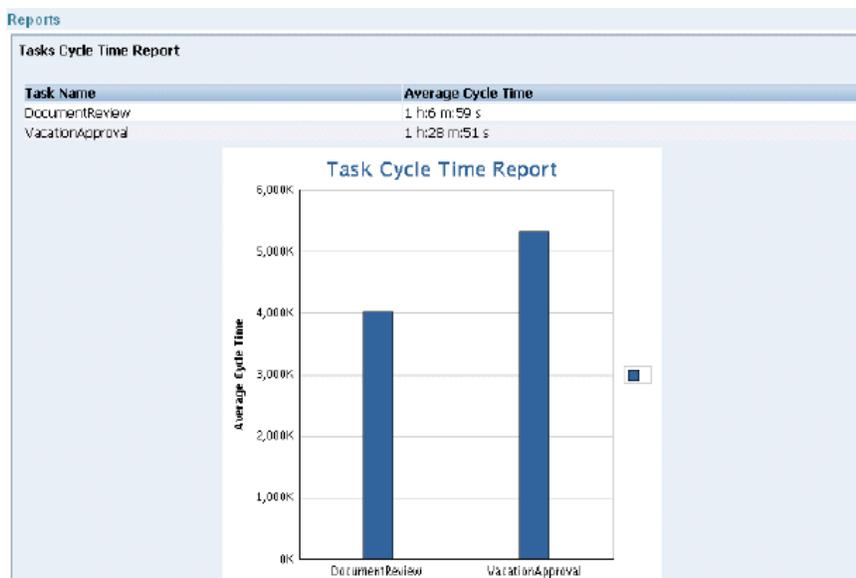


The report shows that the California group, the Supervisor group, and the LoanAgentGroup each have 16 tasks of normal priority. The users rsteven and jcooper have 5 and 22 tasks, respectively, all normal priority. Priorities (highest, high, normal, low, lowest) are distinguished by different colors in the bar chart.

33.7.2.3 Tasks Cycle Time Report

Figure 33–41 shows an example of a Tasks Cycle Time Report.

Figure 33–41 Tasks Cycle Time Report

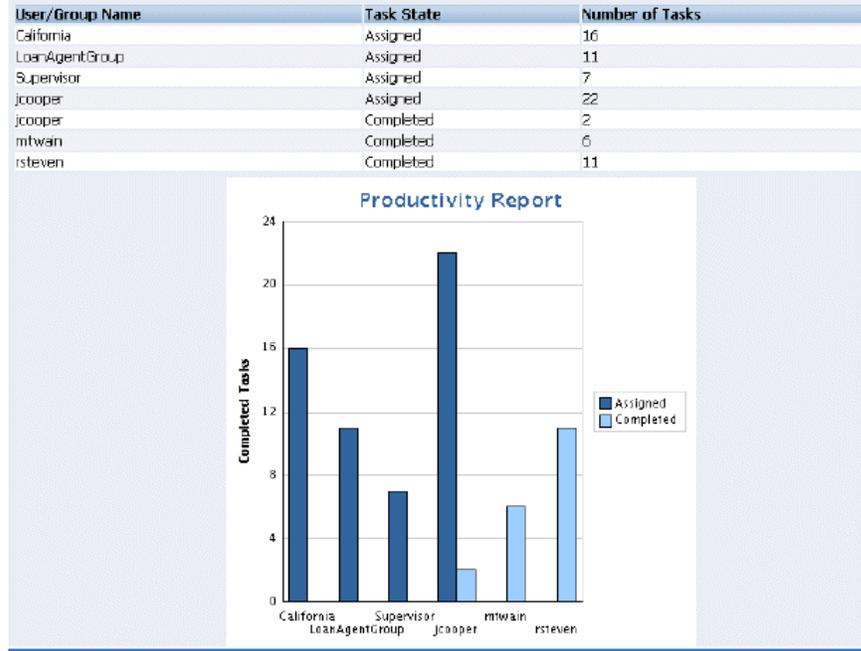


The report shows that it takes 1 hour and 6 minutes on average to complete DocumentReview tasks, and 1 hour and 28 minutes on average to complete VacationApproval tasks. The bar chart shows the average cycle time in milliseconds.

33.7.2.4 Tasks Productivity Report

Figure 33–42 shows an example of a Tasks Productivity Report.

Figure 33–42 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.

33.7.2.5 Tasks Time Distribution Report

Reviewers—Can someone provide a sample report?

33.8 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 screens 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 33.6.2, "How to Set the Worklist Display \(Application Preferences\),"](#) for more information.

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 28, "Human Task 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 28, "Human Task Services,"](#) for information on how to specify message bundles so that custom actions and task titles are displayed in the preferred language.

33.8.1 How to Change the Language Used in the Worklist

The following instructions are based on extracting a user's preferred language from a JAZN XML file.

To change the language:

1. Open the following file:

```
SOA_Oracle_Home\bpel\system\services\config\demo-users-properties.xml
```

2. Change the bits in bold to set the user's preferred language.

```
<languagePreference>en_US</languagePreference>
```

Oracle BPM Worklist supports the locales shown in [Table 33–5](#).

Table 33–5 Languages and Java Locales Supported by Oracle BPM Worklist

Language	Java Locale
English	(en)
English (United States)	(en_US)
German	(de)
Spanish (International)	(es)
Spanish (Spain)	(es_ES)
French	(fr)
French (Canada)	(fr_CA)
Italian	(it)
Japanese	(ja)
Korean	(ko)
Portuguese	(pt)
Portuguese (Brazil)	(pt_BR)
Chinese (Simplified)	(zh_CN)
Chinese (Traditional)	(zh_TW)

33.8.2 How to Change the Time Zone Used in the Worklist

The following instructions are based on extracting a user's time zone from a JAZN XML file.

To change the time zone:

1. Open the following file:

```
SOA_Oracle_Home\bpel\system\services\config\demo-users-properties.xml
```

2. Change the string in bold to set the user's preferred time zone.

```
<timeZone>America/Los_Angeles</timeZone>
```

Reviewers—Do we need to doc where users look up the allowable values?

33.9 Summary

This chapter describes how to access tasks, view task details, and perform actions on the tasks in the sample Oracle BPM Worklist. It also discusses how you can create and share custom views, manage user and group rules, customize task display settings, and perform administrative tasks such as flex field mapping and application customization.

Human Task and Approval Management Integration

This chapter describes Approval Management Service (AMX) integration with the human task services of Oracle SOA Suite. AMX replaces the E-Business Suite (EBS) R12 Approvals Management Engine (AME) and the PeopleSoft 8.48 Approval Workflow Engine (AWE) of previous releases. AMX provides a unified method for defining and controlling approval tasks. You define the approval task in the Human Task Editor of Oracle JDeveloper, and associate the task with a BPEL process.

This chapter contains the following topics:

- [Section 30.1, "Introduction to AMX"](#)
- [Section 30.2, "Designing AMX Approval Tasks in Oracle JDeveloper"](#)

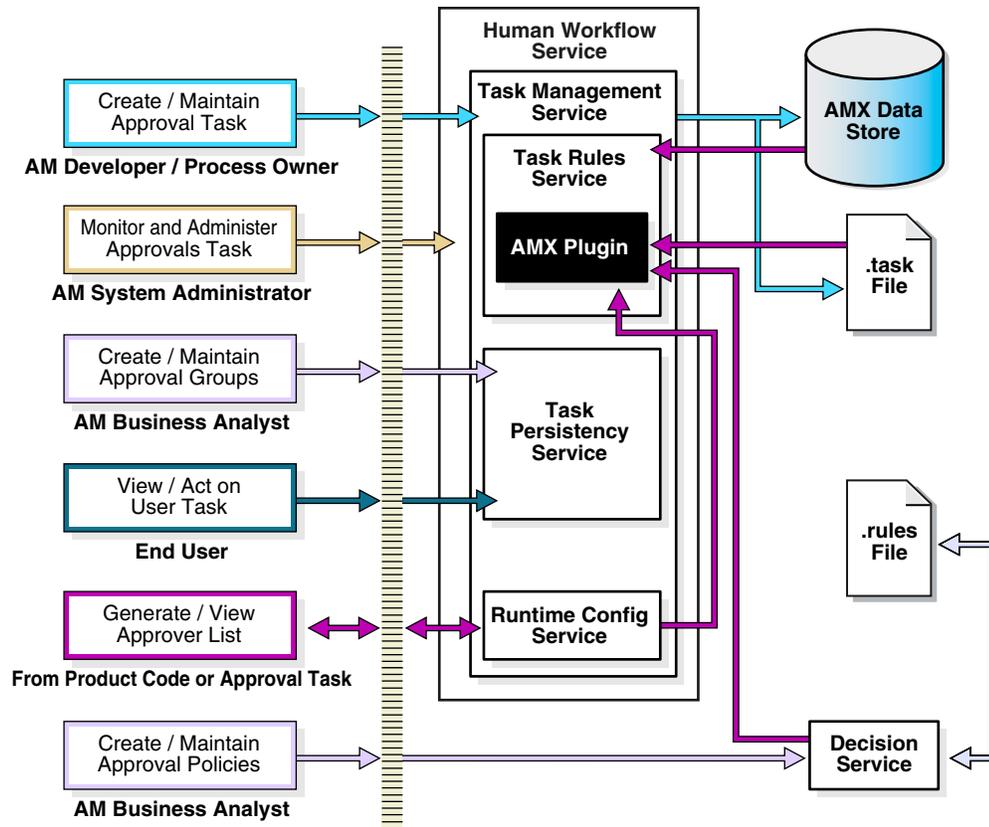
30.1 Introduction to AMX

AMX provides the following key features:

- A unified way to define and control approval management processes
- Tight integration with human workflow services. This provides for easy integration of AMX with a business process.
- Ability to work with a static approver list or have the approver list regenerated each time
- An intuitive method for defining serial and parallel approval paths
- Graphical representation of the approver list, which is easy to read and understand
- Intuitive user interface for a business analyst to use with ease
- Capability to define different notifications for different participant types
- Reuse of existing contexts to get attribute values or terms. These terms can then be used to define conditions or criteria. You define rules with these conditions based on your business needs.
- Terms are inferred from view object (VO) definitions, which are preseeded.
- Approver list builders work like AME handlers to build the approver list.
- Ability to decide which approver list builders to use for your approval process.

[Figure 30-1](#) shows the key AMX and human task integration components. These components are described in subsequent sections of this chapter.

Figure 30–1 Overall Architecture



This section contains the following topics:

- [Section 30.1.1, "AMX Components"](#)
- [Section 30.1.2, "Human Task and AMX Concepts,"](#)
- [Section 30.1.3, "Decision Service and Oracle Business Rules Concepts"](#)

See Also:

- [Chapter 26, "Designing Human Tasks"](#)
- [Chapter 28, "Human Task Services"](#)

30.1.1 AMX Components

Note: For Applications Drop 4, the status of AMX components is as follows:

- Decision service and Oracle business rules are not currently integrated with AMX.
 - HR hierarchies are not available within the identity service. A supervisory hierarchy has been seeded in the identity service and is available for use.
 - Only the Oracle JDeveloper-based AMX setup UI is available.
 - The AMX monitor, AMX admin, and AMX test workbench are not currently available.
 - The default worklist application is supported.
-
-

AMX consists of the following main components:

- Human task service — Provides a method for adding a predefined approval task service to a process flow from the **Component Palette** of Oracle JDeveloper. This service also provides a graphical worklist application to securely act on approvals currently assigned to the user.
- Decision service and Oracle business rules — Evaluate rules and identify the approval list builders that must be implemented for a transaction.
- Identity service — Provides a single source for verifying the existence of all types of participants.

Note: The identity service, decision service, and Oracle business rules are all used together to process rules and validate approvers.

- AMX setup user interface — Provides a graphical user interface for defining and modifying approval taxonomy and rules.
- Worklist application — Provides a Web interface that enables users to act on their assigned tasks.
- AMX monitor — Provides the ability to monitor current flows and view history and audit approvals for previous transactions.
- AMX admin — Provides the administrator with a graphical user interface to configure the approval process and monitor and control the initiated approval process.
- AMX test workbench — Provides methods to run simulation tests for specific approval flows.

Note: The AMX test workbench, worklist application, and AMX monitor provide business analysts with graphical user interfaces for performing their responsibilities.

- Oracle JDeveloper and AMX setup graphical components — Provide the design time user interfaces for creating approval processes.

30.1.2 Human Task and AMX Concepts

This section provides an overview of human task and AMX concepts.

30.1.2.1 Approval Task

An approval task is a special human task that handles approvals. A different approval task is created for each approval requirement based on the business served by it. This is similar to the AME transaction type (for example, an approve new expense report task, approve new purchase order task, and so on).

30.1.2.2 Approver Groups

Approver groups are statically or dynamically generated groups of approvers. Approver groups typically represent functional or subject matter experts outside the transaction's managerial chain of authority (such as human resources or legal counsel) that must approve the transaction before or after management approval.

Approver groups also simulate a forced hierarchy across multiple types of approvers or chain approvers across multiple approver types.

Static approval groups are groups with named members and correspond to a routing slip pattern. The dynamic approval group is treated as an assignment service Java plug-in.

30.1.2.3 Approver List

The approval list is the list of participants generated by AMX for a transaction defined within the approval task. The list is based on the approval policies defined for the approval task.

30.1.2.4 Chain

A chain is a subpath of an approval list builder. This is used in a dual chain approver list builder and an approval group list builder. In the dual chain approver list builder, the approval goes up two separate job level hierarchies. In the approval group list builder, if multiple groups are to be included for a stage, then each group is processed as a separate chain. These chains are always processed in parallel.

30.1.2.5 Configuration Variables

Configuration variables control the behavior of AMX. They are defined for each approval service. Configuration parameters are typically defined when you define the task, such as push back, escalate, and expire. They can be set to a static value or Java code can be executed to identify the value at run time. A parameter also determines whether to execute this code once for the service or every time this configuration value must be checked.

Note: For Applications Drop 4, only static variables are supported.

For example, one variable dictates whether the approver list generation is static or dynamic. If static, AMX must only evaluate the rules to determine the applicable approval list builders for each stage and use them to build the approver list once.

The following sections describe the available configuration variables:

- [Section 30.1.2.5.1, "Administrative Configurations"](#)
- [Section 30.1.2.5.2, "Approval Configurations"](#)

- [Section 30.1.2.5.3, "Workflow Configurations"](#)

30.1.2.5.1 Administrative Configurations The control of administrative behavior and value of this configuration variable can come from a higher layer (for example, the enterprise manager).

[Table 30–1](#) describes the available administrative configuration variable.

Table 30–1 Administrative Configuration Variable

Name	Description	Value
purgeFrequency	Frequency to purge	Positive integers — Specify the frequency in the number of days

30.1.2.5.2 Approval Configurations These configuration variables impact the way in which you build the approver list.

Note: For Applications Drop 4, the approval configuration variables shown in [Table 30–2](#) are not supported.

[Table 30–2](#) describes the available approval configuration variables.

Table 30–2 Approval Configuration Variables

Name	Description	Values
repeatedApprovers	For aggregate notifications	<ul style="list-style-type: none"> ■ OncePerApproval — This is per approval ■ OncePerStage — This is per stage ■ OncePerChain — This is per chain ■ OncePerOccurrence — No aggregation
allowSkippingRuleGeneratedApprover	Allows authorized users to remove approvers based on the definitions of Oracle business rules (versus resulting from adhoc insertions)	<ul style="list-style-type: none"> ■ True — Allow skip ■ False — Disallow skip
allowSelfApproval	Allows a requester to be the final approver and the only approver within this list builder. The starting point must be null. Otherwise, it may consider that the list builder wants to be excluded from the self approval behavior.	<ul style="list-style-type: none"> ■ True — Allows (Automatically approved if the requirement is met. The starting point is set to requester). ■ False — Disallow (Starting point is set to the requester's manager).
rejectionResponse	The behavior when someone rejects on the subordinate dimension. If it is a header dimension, only StopAll is applicable.	<ul style="list-style-type: none"> ■ StopLine — Continues all except the particular instance that was rejected. ■ StopStage — Continues all other parallel stages. ■ StopAll — Stops the approval and returns a reject outcome.

30.1.2.5.3 Workflow Configurations

Table 30–3 describes the available workflow configuration variables.

Note: The following features appear in the AMX setup user interface, but are not supported for Applications Drop 4:

- Pushback
- Reassign
- Escalate

Table 30–3 Workflow Configuration Variables

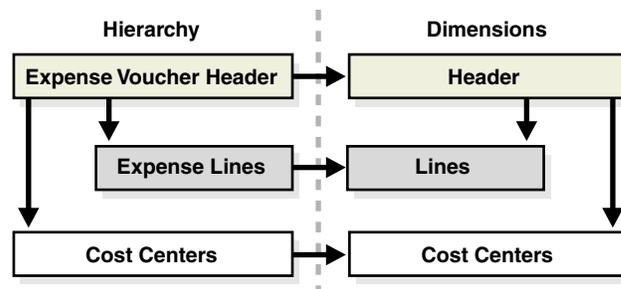
Name	Description	Values
continuousRuleEvaluation	Controls whether or not the routing slip interpreter returns to the task rule service for the next set of approvers at every possible point	<p>Note: For Applications Drop 4, only a value of <code>True</code> is supported.</p> <ul style="list-style-type: none"> ■ <code>True</code> — Returns to ask for the routing slip of the next set of approvers ■ <code>False</code> — Requests a full routing slip once
Pushback	Allows previous approvers to redo their decisions	<ul style="list-style-type: none"> ■ <code>True</code> — Allows ■ <code>False</code> — Disallows
Delegate	Allows a different user to decide on behalf of another user	<ul style="list-style-type: none"> ■ <code>True</code> — Allows ■ <code>False</code> — Disallows
Reassign	Allows a user to reassign a task to a different user (chain of authority insertion)	<ul style="list-style-type: none"> ■ <code>True</code> — Allows ■ <code>False</code> — Disallows
Expire	The time at which a task expires	<ul style="list-style-type: none"> ■ <code>Days</code> — Time in days ■ <code>Hours</code> — Time in hours ■ <code>Minutes</code> — Time in minutes
Escalate	The time at which to escalate a task	<ul style="list-style-type: none"> ■ <code>Days</code> — Time in days ■ <code>Hours</code> — Time in hours ■ <code>Minutes</code> — Time in minutes
Escalated Approver	The assigned approver when a task is escalated	This is typically a dynamic value.
Withdraw	Allows the requester or admin to retract an approval	<ul style="list-style-type: none"> ■ <code>True</code> — Allows ■ <code>False</code> — Disallows
Skip	Allows the current approver to be skipped.	<ul style="list-style-type: none"> ■ <code>True</code> — Allows ■ <code>False</code> — Disallows
Adhoc Insertion	Allows for adhoc routing	<ul style="list-style-type: none"> ■ <code>True</code> — Allows ■ <code>False</code> — Disallows
Request Information	Requests for additional information	<ul style="list-style-type: none"> ■ <code>True</code> — Allows ■ <code>False</code> — Disallows
onErrorNotify	Identifies the participant to notify when an error occurs	The participant

30.1.2.6 Dimensions (or Message Attribute Group)

Dimensions categorize the rule context. Dimensions normally have a one-to-one mapping to collection terms defined in Oracle business rules. For instance, expense vouchers can have hierarchical groupings of header, lines, and cost centers. For approvals, the dimensions defined can be header, lines, and cost centers.

It is not necessary for all dimensions to be mapped to a stage. If rules are not necessary for the dimension, then dimensions do not need to be mapped to a stage. In the human task context, this is referred to as the message attribute group. [Figure 30–2](#) provides an overview.

Figure 30–2 Entity Hierarchy Being Mapped to Dimensions



See Also: [Section 30.2.3, "Create a Dimension"](#)

30.1.2.7 Human Task Editor

This editor enables you to specify task settings such as task outcomes, payload structure, task participants, assignment and routing policy, expiration and escalation policy, notification settings, and so on.

See Also: [Section 26.6, "Creating the Human Task Definition with the Human Task Editor"](#)

30.1.2.8 List Builders

List builders abstract the specification of similar types of actions. An approver list builder moves up a hierarchy and, based on the list builder constraints, returns an approver list. The following approver list builders are supported:

- Supervisory — Ascends the primary supervisory hierarchy starting at the requester or at a given approver, generating a chain that has a fixed number of approvers in it
- Job level — Ascends the supervisory hierarchy starting at a given approver and continuing until an approver with a sufficient job level is found.
- Position — Ascends the position hierarchy starting at the requester's or at a given approver's position and going up a specified number of levels or to a specific position.
- Dual chain — Ascends two separate job level hierarchies
- Approval group — Includes a predefined approver group in the approver list. This group can be a static or dynamic group.

See Also: [Section 30.2.5, "Configure Approval List Builders for a Stage"](#)

30.1.2.9 List Builder Usage

Note: Since Oracle business rules integration is not available with Applications Drop 4, the action configurations and constraints are currently defined with list builder usage.

Approval list builder usage defines how an approval list builder is used for a stage within an approval service. It defines the following:

- Preference — Specifies the order in which to process the list builder for a stage
- Default voting regime — Specifies the voting behavior to follow

30.1.2.10 Notification

Notification refers to the e-mail, short message service (SMS), or instant message (IM) channel that notifies a user or role that a certain task requires their attention.

See Also:

- [Chapter 23, "Notifications and the Oracle User Messaging Service"](#)
- [Section 26.6.7, "Specifying Participant Notification Preferences"](#)
- [Section 28.2, "Notifications from Human Workflow"](#)

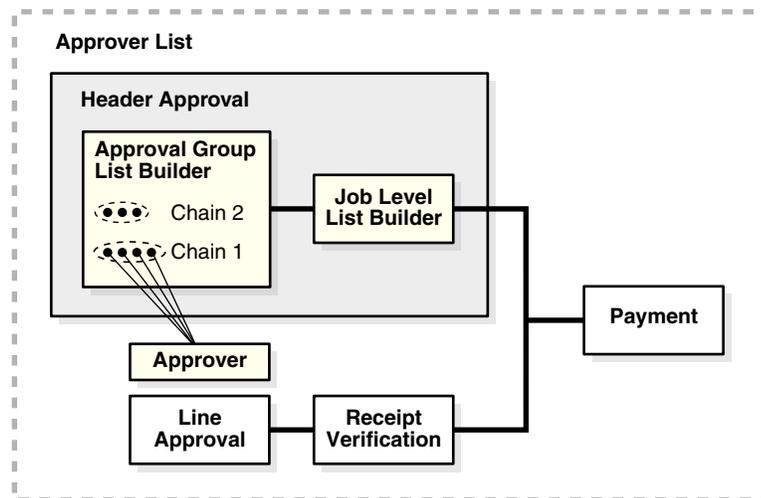
30.1.2.11 Parts of an Approver List

An approver list can be broken into the following parts:

- Stages
- List builders
- Chains
- Approver

The shape of the approver list is defined while defining the approval structure. [Figure 30–3](#) provides an overview.

Figure 30–3 Approver List Structure



30.1.2.12 Routing Slip

The routing slip definition dynamically routes tasks to multiple users. It includes information such as workflow sequence, assignment policy, notification policy, callback policy, escalation policy, renewal policy, and so on.

See Also: [Chapter 26, "Designing Human Tasks"](#)

30.1.2.13 Stages

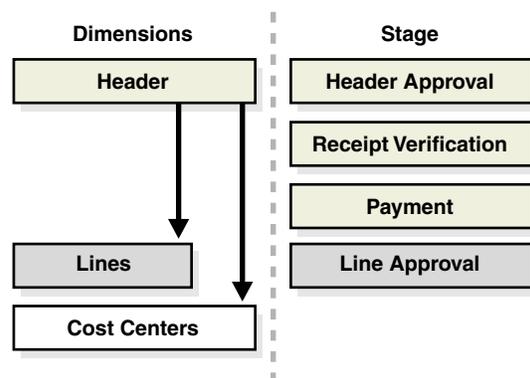
A stage is a specific approval point associated with a dimension. However, a dimension can be associated with multiple approval stages. A set of rules or approval policies are associated with each stage. A separate list of approvers is generated for each stage, based on the rules applicable and the approver list builders defined in the rule.

For example, an expense report can have the following stages defined:

- Header approval stage associated with the header dimension
- Line approval stage associated with the lines dimension
- Receipt verification stage associated with the header dimension
- Payment stage associated with the header dimension

Stages can be modeled to be serial or parallel with respect to each other. This is performed while defining the approval structure. [Figure 30–4](#) provides an overview.

Figure 30–4 *Dimensions Being Mapped to Stage*



See Also: [Section 30.2.4, "Create the Stages"](#)

30.1.2.14 .task file

The `.task` file is the metadata task configuration file that stores the task settings specified with the Human Task Editor in Oracle JDeveloper.

See Also:

- [Section 30.2, "Designing AMX Approval Tasks in Oracle JDeveloper"](#)
- [Section 26.6, "Creating the Human Task Definition with the Human Task Editor"](#)

30.1.2.15 User Task

The user task refers to a work item or some task that requires manual intervention. This was referred to as notification in previous releases of Oracle Workflow (OWF).

30.1.2.16 Worklist Application

The worklist application interacts with the workflow service to retrieve user tasks and enable users to perform actions on their tasks.

See Also: [Part IV, "Human Workflow Service Component"](#) for chapters on using the worklist application

30.1.3 Decision Service and Oracle Business Rules Concepts

This section provides an overview of decision service and Oracle business rules concepts.

Note: These features are not available with Applications Drop 4.

30.1.3.1 Approval Routing Policies

Approval routing policies or rules determine the approver list builders to use to identify the approvers or FYI notification recipients required for a business transaction.

In some cases, routing policies are seeded for out-of-the-box approval capability. However, these are typically defined by the customer, since approval policies vary between organizations.

Each routing policy defines the following details:

- Approval policy type
- Action
- Criteria
- Priority
- Response type
- Validity period

30.1.3.2 Approval Policy Type

Note: List modification and substitution actions are not available with Applications Drop 4.

This type describes the specific action being performed on the approver list. The policy types supported are:

- List creation
- List modification
- Approver substitution

30.1.3.3 Action

Based on the approval policy type, different information must be defined for each action. These are also known as the action configurations.

List creation policy instructs AMX to create the approver list based on the following criteria:

- Approver list builder to be used
- Stopping rule
- Response type
- All members included in the list
- Voting regime for the rule (only for the approval group list builder)

List modification policy instructs AMX to modify the approver list based on the following criteria:

- Action (for extend list or truncate list)
- List builder to use (for extend list)
- Stopping rule for the action (for extend list)

Approver substitution policy instructs AMX to replace one approver in the approver list with another (if present in the list).

30.1.3.4 Conditions

Conditions in Oracle business rules are Boolean expressions that compare a term to other terms or constant values. Conditions can also be created as standalone entities for re-use in other rules. The operators used in conditions are defined in metadata and can be extended by application developers.

30.1.3.5 Criteria

The criteria is the set of conditions under which a rule is applicable. If all the conditions in a routing policy are true, then that routing policy is active for the transaction.

30.1.3.6 Priority

Rule arbitration is carried out by means of prioritizing rules. This is performed in Oracle business rules.

30.1.3.7 Response Type

This type describes the specific response expected from the approver for the approval task. The various responses allowed are:

- Required (for approval notification)
- Not required (FYI)
- Optional (for review). This response is not available in Applications Drop 4.

30.1.3.8 Rule Context

Rule context is the entity being considered for approval. It can be a purchase order, an expense report, or any other document. It typically points to the VO created to define the terms and context in Oracle business rules. The VO is the business component for Java (BC4J) object used to create the rule context. The shape of a given rule context

provides an execution model for rule evaluation, depending on which items from the rule context are used to define a given rule.

30.1.3.9 Terms

Terms are semantic units of information relevant to a business user. Most terms originate from member fields of business objects, such as the first name attribute of a person.

The decision service does not maintain its own library of terms. Instead, the decision service uses the existing Siebel Business Analytics (SBA) and BC4J layers. A term is a data element from the BC4J or SBA layers that has been designated for use either in one or more contexts, rules, conditions, or actions.

The decision service uses the Oracle Fusion Middleware BC4J as a data source. The Oracle Fusion Middleware BC4J stack is used for accessing the intratransactional and transient data present during the online application runtime. BC4J is also used as a source of persistent data, initially through the BC4J APIs. Once SBA delivers sufficient BC4J integrations, offline BC4J entity objects (EOs) are accessed through the SBA layer instead of directly through the BC4J APIs. This allows for more efficient correlation to other data sources.

30.1.3.10 Validity Period

The validity period represents the time period for which the rule is valid. This also includes the time stamp.

Rules with the following validity period can be created:

- Rules starting on the current date and at the current time
- Rules starting at a date and time in the future
- Rules ending on the current date and at the current time
- Rules ending at a date and time in the future

The following validity periods are not allowed:

- Rules starting before the current date and time
- Rules ending before the current date and time
- Rules with the ending date and time less than or equal to the starting date and time
- The same rule implemented multiple times with overlapping validity periods

30.2 Designing AMX Approval Tasks in Oracle JDeveloper

This section describes how to design an approval task. You design this task by creating a human task service component and selecting the **Composite Approver** participant type in the Human Task Editor of Oracle JDeveloper. When complete, this task invokes the AMX approval service as an assignment service to retrieve the set of approvers and populates the routing slip.

This section describes the following tasks:

- [Section 30.2.1, "Review Prerequisites"](#)
- [Section 30.2.2, "Select the Composite Approver Participant Type"](#)
- [Section 30.2.3, "Create a Dimension"](#)

- [Section 30.2.4, "Create the Stages"](#)
- [Section 30.2.5, "Configure Approval List Builders for a Stage"](#)

Note: This section *only* describes how to create and configure the composite approver participant type. The composite approver participant type enables you to integrate the AMX approval task with a human task. For instructions on creating and designing all parts of a human task, see [Section 26.6, "Creating the Human Task Definition with the Human Task Editor."](#)

30.2.1 Review Prerequisites

Before designing an approval task in the Human Task Editor, you must satisfy the following prerequisites:

- You must identify the primary business object and create the approval content type. For example, the primary business object can be an expense voucher. Identify and construct the entity hierarchy of this object using infrastructure services such as BC4J. You define this context in the decision service and Oracle business rules, and reference it as the content type within AMX. This entity hierarchy determines the dimensions available within AMX.

Notes:

- VO introspection is not supported with Applications Drop 4.
 - It is important to define as many VO attributes as possible so they do not require extensions.
-
- You must define the payload to pass to the human task. [Table 30–4](#) describes the required contents of the payload.

Table 30–4 Payload Requirements

Name	Description
requestor (maps to the creator in the human task)	The principal that initiated the approval process
requestorRole	The principal role that initiated the approval process
identificationKey	The fully-qualified key to uniquely identify this instance. This key can look up an approval at a later time.
transactionDate (map to rule property later on)	The timestamp to identify the version of rules to use for evaluation
pk{n} (map to context keys later on, n can be 1, 2, 3 and so on)	The primary key for the business entity. These attributes are passed to Oracle business rules.

30.2.2 Select the Composite Approver Participant Type

1. Create a human task service component in which to design the approval task. When you create a human task service component, the Human Task Editor is automatically invoked and a `.task` file is automatically created. This editor can also be invoked at a later time by double-clicking the human task service component in the SOA Composite Editor. See [Section 26.6, "Creating the Human Task Definition with the Human Task Editor"](#) for the following instructions:

- Adding a human task service component to a SOA composite application
 - Invoking the Human Task Editor to design the approval task
2. Enter a name in the **Title** field (for example, `AMX Generated Approvers` can be entered).
 3. Click the **Add Participant Type** icon in the **Assignment and Routing Policy** section of the Human Task Editor.



The Add Participant Type window appears.

4. Enter the following details.

Field	Description
Type	Select Composite Approver . This refreshes the window to display new fields.
View Object Definition	Specify the complete path for the defined VO (for example, <code>oracle.tip.tools.ide.workflow.editor.purchaseOrderVO</code> can be specified). Note: Be aware of the following issues for Applications Drop 4: <ul style="list-style-type: none"> ■ The Browse icon is disabled. ■ The VOs are not used by AMX for validation.
List Substitution Policy	Note: For Applications Drop 4, The Create icon is disabled and list substitution policies are not supported. List substitution type policies enable you to substitute approvers on the list with other approvers. For example, you can substitute one supervisor in the approval hierarchy with a second supervisor if the first approver is on vacation.
List Modification Policy	Note: For Applications Drop 4, The Create icon is disabled and list modification policies are not supported. List modification type policies allow for the modification of the approver list generated by the applicable policies calculated using the list creation rules. For example, you can extend the approval hierarchy to additional levels or truncate the approval hierarchy after a specific approver has acted.

When complete, the **General** section of the Add Participant Type page appears as follows:

Add Participant Type

General

Type: Composite Approver

View Object Definition: oracle.tip.tools.ide.workflow.editor.purchaseOrderVO

List Substitution Policy:

List Modification Policy:

Dimensions

Name	Key List

Stages

```

graph TD
    Start --> NoStages[No Stages Defined]
    NoStages --> End
  
```

Properties

Allow Self Approval: true

30.2.3 Create a Dimension

You now create a dimension. Dimensions categorize the rule context. The first dimension you create is usually based on the view object defined in the **View Object Definition** field of the Add Participant Type window. For Applications Drop 4, this feature is not available. For all subsequent dimensions that you create, view links must be entered.

1. Click the **Create** icon in the **Dimensions** section of the Add Participant Type window.

The Add Dimension window appears.

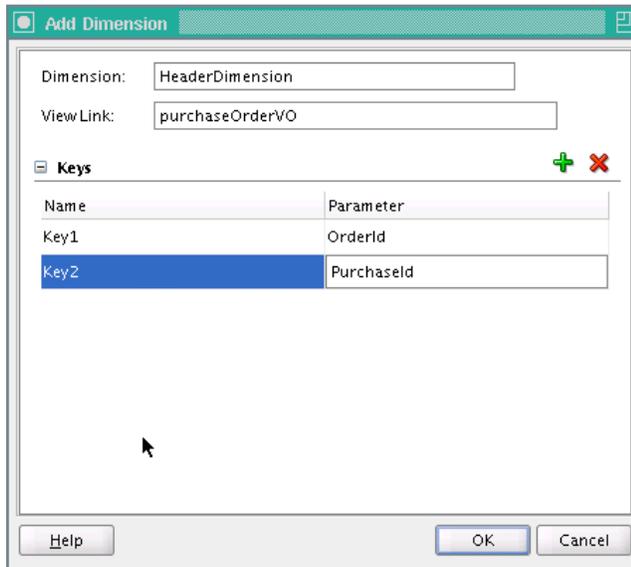
2. Enter the following details.

Field	Description
Dimension	Enter a dimension name (for example, <code>HeaderDimension</code> can be entered).
View Link	Enter the complete path of the view link (VL) (for example, <code>purchaseOrderVO</code> can be specified). This VL should be linked to the main VO. There must be only one main VO. The other dimensions are linked to the main object by VLs. Note: In Applications Drop 4, the VO/VL information is not used at runtime, so no validation is performed here.

3. Click the **Create** icon in the **Keys** section.
4. Enter a value in the **Parameter** field for **key1**. For example, `orderId` can be entered.

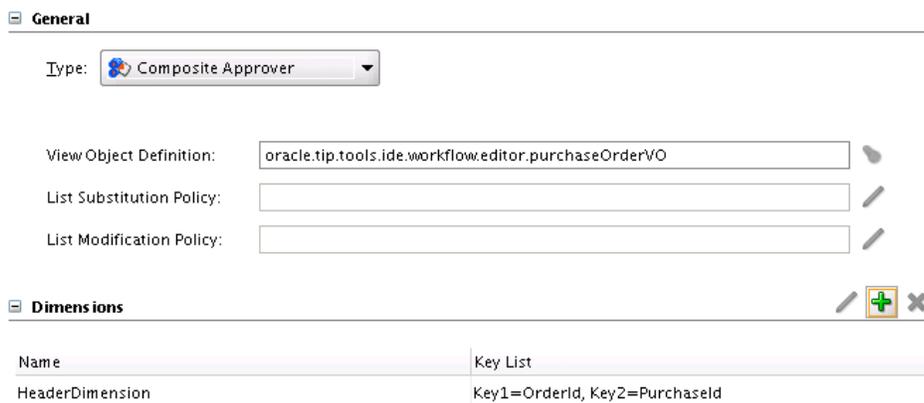
5. Click the **Create** icon in the **Keys** section again.
6. Enter a value in the **Parameter** field for **key2**. For example, `PurchaseId` can be entered.

These mapping keys link the primary keys passed in the payload to the terms in the VO/VL for this dimension. When complete, the window appears as follows.



7. Click **OK**.

You are returned to the Add Participant Type window. The **Dimensions** section displays the following information.



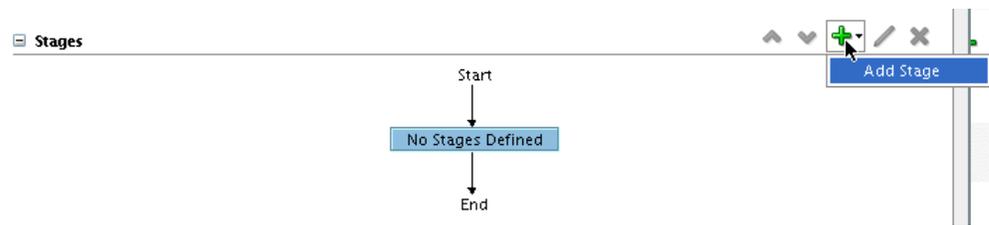
See Also: [Section 30.1.2.6, "Dimensions \(or Message Attribute Group\)"](#)

30.2.4 Create the Stages

You now create and associate stages with the dimension that is processed in it. A set of rules or approval policies are associated with each stage. A stage is always associated with one and only one dimension. However, one dimension can be associated with more than one stage. For example, an expense header can be completed in three stages: header approval, receipt verification, and payment.

1. Select **No Stage Defined**.

- Click the **Create** icon in the **Stages** section of the Add Participant Type window.

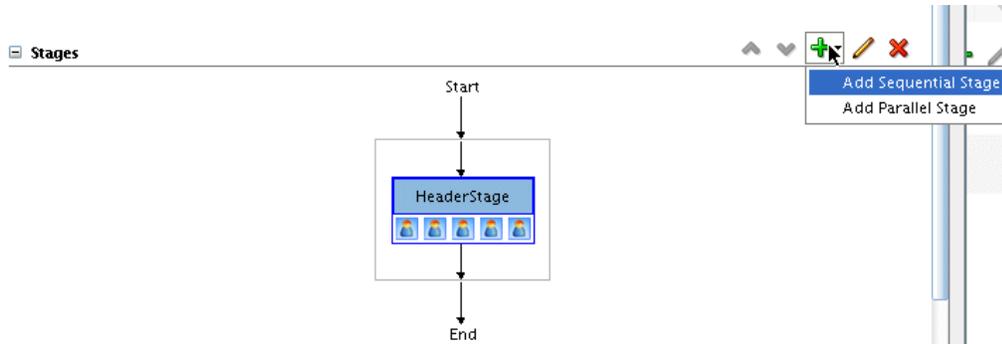


- Select **Add Stage** from the dropdown list.
The Add Stage window appears.
- Enter the following details.

Field	Description
Stage	Enter a name (for example, <code>HeaderStage</code> can be entered).
Dimension	Select a dimension with which to associate this stage. You created this dimension in Section 30.2.3, "Create a Dimension."
Continuous Evaluation	<p>Note: This feature is not supported in Applications Drop 4.</p> <p>This field is for specifying if the routing slip interpreter must return to the task rule service for the next set of approvers at every possible point.</p> <ul style="list-style-type: none"> True — Returns to ask for the routing slip of the next set of approvers False — Requests a full routing slip only once

When complete, the window appears as follows:

- Click **OK**.
- Select the stage (for example, **HeaderStage** can be selected).
- Click the **Create** icon in the **Stages** section again.
- Select **Add Parallel Stage** from the dropdown list. This creates a stage in parallel with the one created in Step 4.

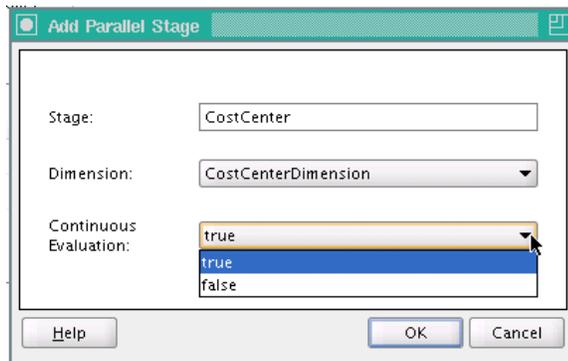


The Add Parallel Stage window appears.

9. Enter the following details.

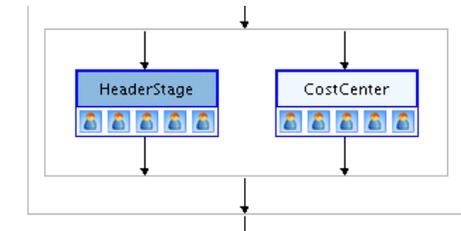
Field	Description
Stage	Enter a name (for example, <i>CostCenter</i> can be entered).
Dimension	Select a dimension with which to associate this stage.
Continuous Evaluation	This feature is not supported for Applications Drop 4.

When complete, the window appears as follows:



10. Click **OK**.

The **Stages** section displays both stages in parallel to one another.

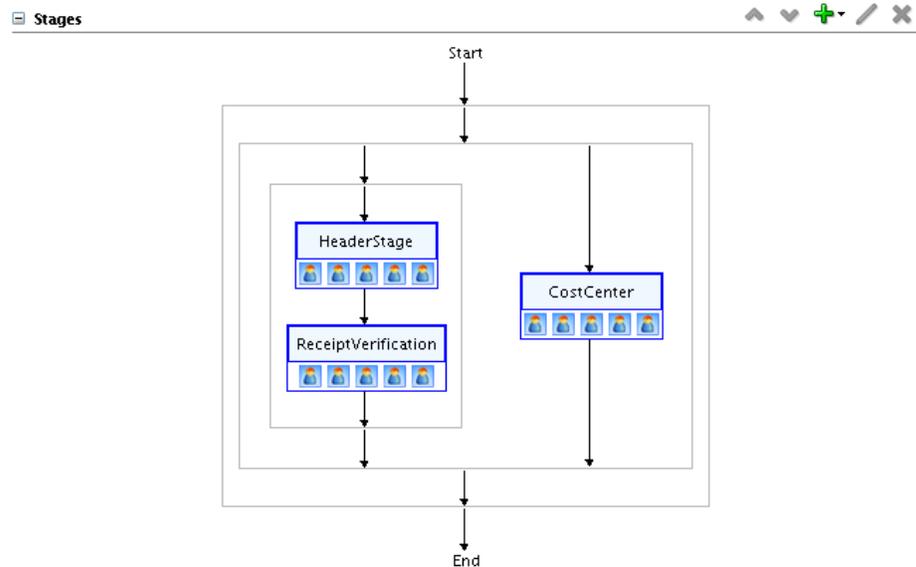


11. Select a stage under which to create a sequential stage. (for example, **HeaderStage** can be selected).
12. Click the **Create** icon in the **Stages** section.
13. Select **Add Sequential Stage** from the dropdown list. This creates a new stage immediately *below* the stage selected in Step 11.

The Add Sequential Stage window appears.

14. Enter details similar to those entered in Step 9 (for example, ReceiptVerification can be entered as the stage name), and click **OK**.

The **Stages** section appears as follows:



Note the **^** and **v** icons above the stages. These icons enable you to change the order of stages. For this example, assume you select the lower stage (**ReceiptVerification**) and click the **^** icon. This moves the lower stage (**ReceiptVerification**) above the upper stage (**HeaderStage**). As another example, assume you select the new upper stage (**ReceiptVerification**) and click the **v** icon. This moves **ReceiptVerification** back below **HeaderStage**.

See Also: [Section 30.1.2.13, "Stages"](#)

30.2.5 Configure Approval List Builders for a Stage

You now define approval steps using the available approver list builders. Approval list builder usage defines how an approval list builder is used for a stage within an approval service.

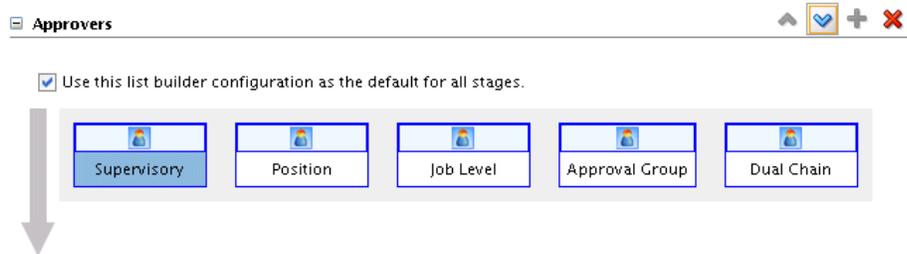
This section contains the following topics:

- [Section 30.2.5.1, "Arranging the Order of Approval List Builders"](#)
- [Section 30.2.5.2, "Deleting Approval List Builders"](#)
- [Section 30.2.5.3, "Adding Approval List Builders"](#)
- [Section 30.2.5.4, "Adding Approval List Builder Policy Actions"](#)

30.2.5.1 Arranging the Order of Approval List Builders

You can arrange approval list builders in either serial or parallel order. The serial and parallel orders govern the order in which approvers identified by the stage receive notifications and are assigned to the approval task. By changing the serial and parallel order, users can change the order of approvals. For example, some companies need management approvals performed before receipt verification. In another company, employees must first get their receipts verified from the payables team and then seek management approval.

1. Double-click a stage (for example, **HeaderStage** can be selected).
 This displays the default approver list builders included with each stage. By default, these list builders are executed in parallel.

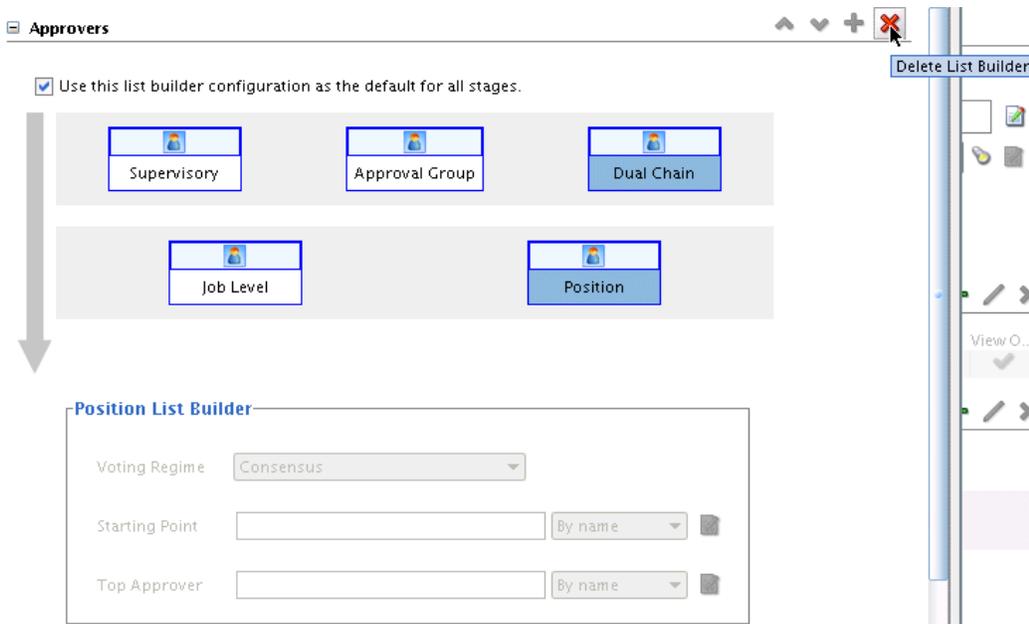


2. Change the order of the approver list builders from parallel to serial. For this example, assume you select the **Job Level** and **Position** approval list builders and click the **v** icon to move both builders below the **Supervisory**, **Approval Group**, and **Dual Chain** approval list builders. This configures the list builders to be executed in serial order, instead of parallel.

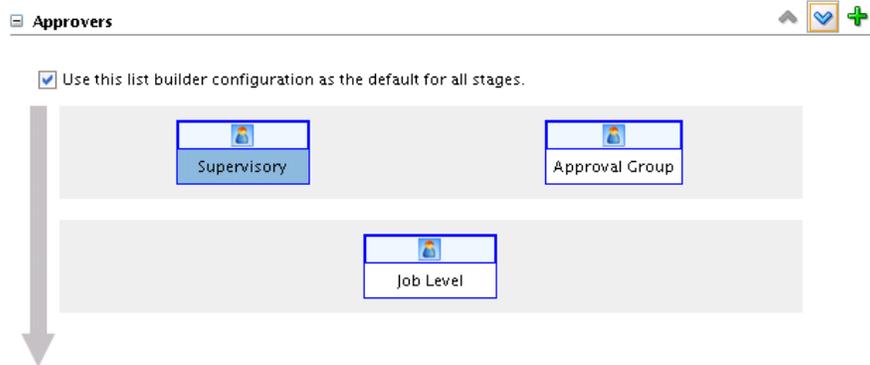
30.2.5.2 Deleting Approval List Builders

You can delete approval list builders that are not required.

1. Delete two of the approval list builders. For this example, assume you select the **Dual Chain** and **Position** approval list builders and click **Delete**. These builders are not required for this example.



When complete, the window appears as follows:



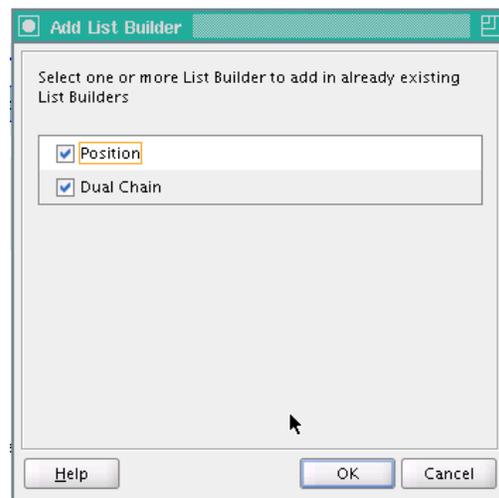
Note: Use of the **Create** icon of the **Approval Policy** field is not supported for Applications Drop 4. In a future Applications Drop, this link will provide access to Oracle business rules.

2. Select and delete additional approval list builders as necessary (for example, the **Approval Group** and **Job Level** approval list builders can be selected).

30.2.5.3 Adding Approval List Builders

You can add approval list builders.

1. Click the **Create** icon.
The Add List Builder Dialog window appears.
2. Select the approval list builders to add to the current builders. For this example, the **Position** and **Dual Chain** check boxes can be selected.



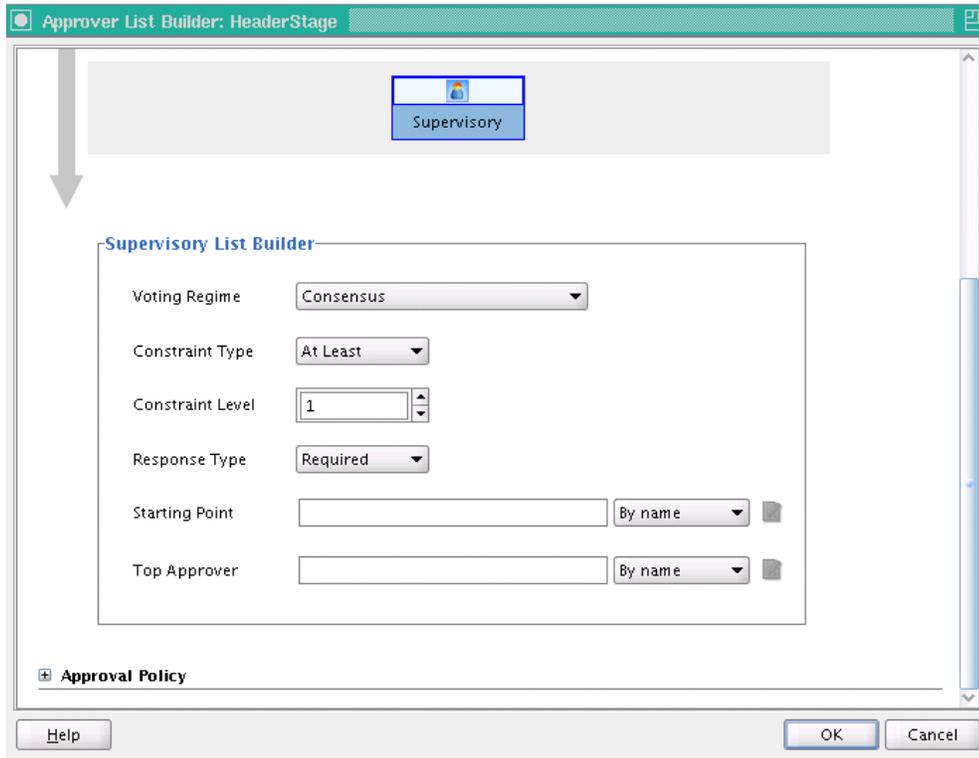
3. Click **OK**.

30.2.5.4 Adding Approval List Builder Policy Actions

You can specific policy actions for each approval list builder.

1. Double-click the **Supervisory** approval list builder.

The Supervisory List Builder window appears. This enables you to specify the approval policy actions.

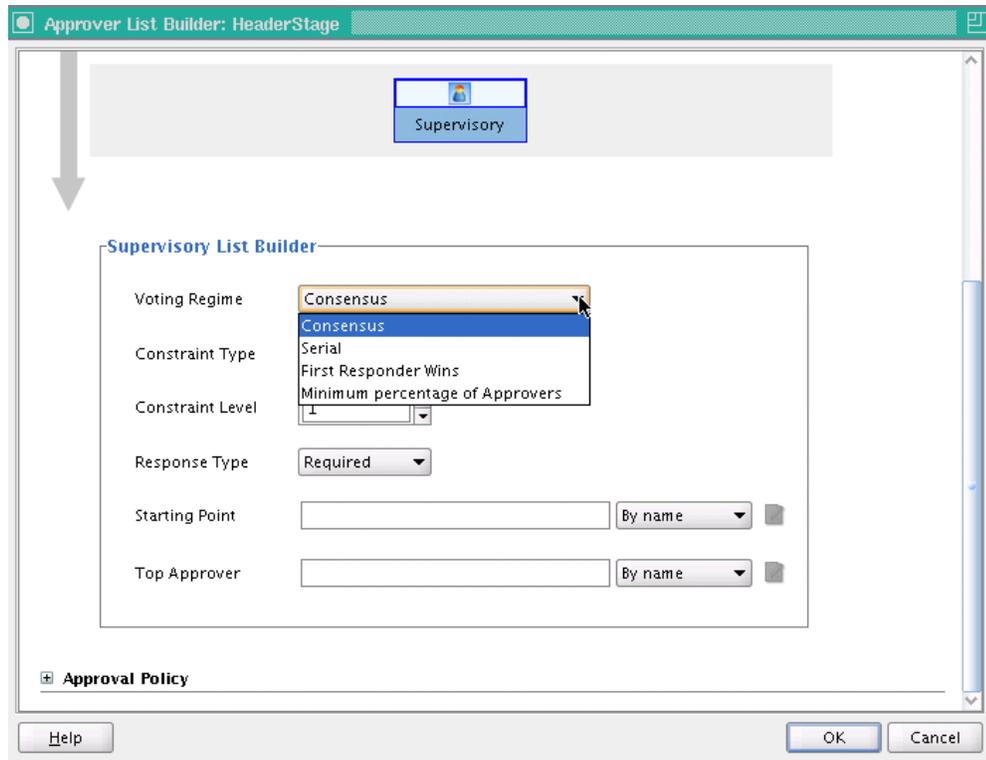


2. Enter the following details:

Field	Description
Voting Regime	<p>Select a voting behavior to follow. Voting is a method of decision making that enables a group to determine its opinion. For example, Minimum percentage of Approvers can be selected. This selection refreshes the window to display a second list, from which 39 is selected.</p> <p>This selection enables the task to be computed as soon as the minimum percentage is met. For example, assume four users are assigned to act on a task, the default outcome is APPROVE, and the minimum 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 already been satisfied.</p> <p>The Voting Regime list also offers additional selections:</p> <ul style="list-style-type: none"> ■ Consensus — Select to process an approval in parallel and require all participants to respond. ■ Serial — Select to force the approvers to act serially in the order defined in the approver list. For example, assume users Joe and Mike both have sequence 1. If user Joe comes first, that user is notified prior to user Mike. ■ First Responder Wins — Select to process an approval in parallel but only take the first response.

Field	Description
Constraint Type	<p>Select a voting constraint type. The values that appear are based upon the type of approval list builder selected.</p> <ul style="list-style-type: none"> ▪ At Most or At Least (displays for Supervisory, Job Level, Dual Chain, and Position) When climbing a hierarchy with missing levels, this selection decides whether to stop below or above the level in the rule. ▪ Absolute or Relative (displays for Job Level, Dual Chain, and Position) Select to use absolute or relative levels. ▪ Level (displays for Supervisory, Job Level, Dual Chain, and Position) Enter a positive integer to indicate the number of steps. ▪ Chain Number (displays for Dual Chain) Specify the chain number. ▪ Group Name (displays for Approval Group) The approval group name.
Constraint Level	Select a voting constraint level.
Response Type	<p>Select a response type.</p> <p>See Also: Section 30.1.3.7, "Response Type" for descriptions of available values</p>
Starting Point	This field is left blank for this example. This field enables you to build an XPath expression for passing data by clicking the XPath Expression Builder icon to the right of this field.
Top Approver	This field is left blank for this example. This field specifies the top level participant in the approval hierarchy. You do not go past the top approver in a hierarchy.

The Supervisory List Builder window displays the following details:



3. Click **OK**.

The Supervisory List Builder window shown in Step 2 contains fields for **Voting Regime**, **Starting Point**, **Top Approver**, and so on. The fields that display can differ based on the approval list builder selected. [Table 30–5](#) identifies and describes the fields that appear for all five approval list builders.

Table 30–5 *Fields of the List Builder Property Window*

List Builder Window Field	Source	Description	Supervisory	Job Level	Position	Dual Chain	Approval Group
Requester	Payload	Requester of this approval as a user.	Yes	Yes	No	No	No
Requester Role	Payload	Requester of this approval as a group.	No	No	Yes	No	No
Allow Self Approval	Global configuration	Indicates if the requester is allowed to approve their own approval as the only and final approver within this list builder. The Starting Point field must be set to the requester. The requester is skipped if they do not have sufficient authority.	Yes	Yes	Yes	No	No
Starting Point	List builder map	The starting point of a list. This field is normally left empty. A value is automatically selected, which in most cases is the requester’s manager.	Yes	Yes	Yes	No	No
Top Approver	List builder map	Indicates to <i>not</i> go beyond this principal in a hierarchy	Yes	Yes	Yes	Yes	No

Table 30–5 (Cont.) Fields of the List Builder Property Window

List Builder Window Field	Source	Description	Supervisory	Job Level	Position	Dual Chain	Approval Group
Include All At Same Level	List builder map	Include all approvers at the same job level	No	Yes	No	Yes	No
Include Approvers	List builder map	Indicates who to include. Values can be All , Final Approver Only , and Manager Final Approver	No	Yes	No	Yes	No
Starting Point Chain 1	List builder map	The starting point for chain one	No	No	No	Yes	No
Starting Point Chain 2	List builder map	The starting point for chain two	No	No	No	Yes	No
Allow Empty Group	List builder map	Allows an empty approval group. If set to True , an exception is not raised if the approval group to insert into the approver list does not have a member (for example, if a dynamic approval group does not return an approver).	No	No	No	No	Yes

AMX task approval configuration is now complete. Proceed with the remaining steps of human task design such as defining notification methods and associating the human task with a BPEL process by following the procedures in [Section 26.6](#), "Creating the Human Task Definition with the Human Task Editor."

See Also: [Section 30.1.2.8](#), "List Builders"

Human Task and Microsoft Excel Integration

This chapter describes how to integrate the desktop application capabilities of Microsoft Excel 2003 with the enterprise system capabilities of Oracle SOA Suite. This type of integration enables you to invoke a BPEL process from Microsoft Excel and attach Microsoft Excel Workbooks to e-mail notifications. This eliminates the need for switching between tools to integrate Microsoft Office with Oracle SOA Suite.

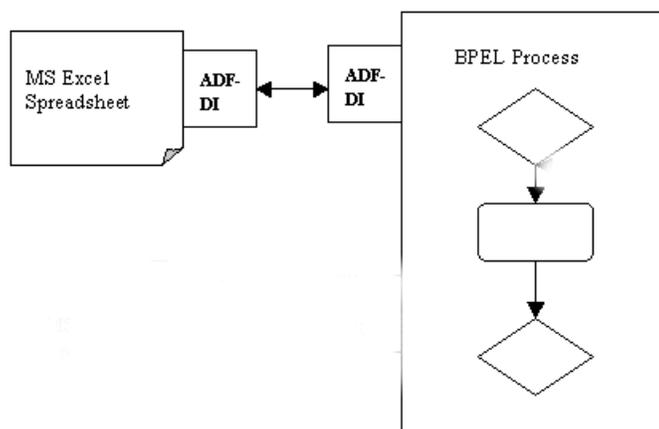
This chapter contains the following topics:

- [Section 31.1, "Invoking a BPEL Process from Excel Workbook"](#)
- [Section 31.2, "Attaching Excel Workbooks to E-mail Notifications"](#)

31.1 Invoking a BPEL Process from Excel Workbook

You can invoke a BPEL process from Excel Workbook. This is achieved through use of an Application Development Framework desktop integration (ADF DI) plug-in installed on the host from which the Excel document invokes the BPEL process. [Figure 31-1](#) provides an overview.

Figure 31-1 *Invoking a BPEL Process from Excel*



- 1.
- 2.

31.2 Attaching Excel Workbooks to E-mail Notifications

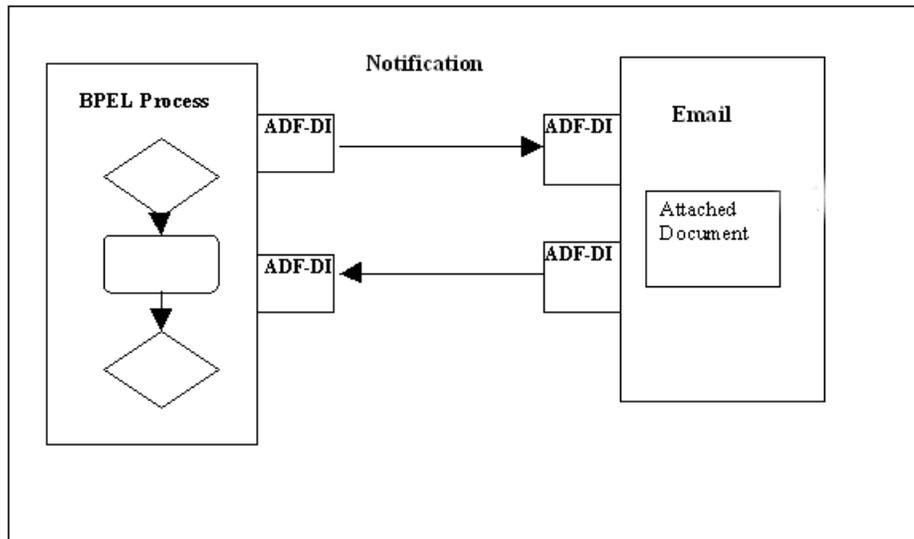
You can attach an Excel Workbook with task details as part of a human task workflow e-mail notification. Excel Workbook provides you with an alternative to the worklist application for viewing current task details and performing necessary actions.

You receive an e-mail notification about a new task with the Excel attachment. When the attachment is opened, you are directed to a login page and prompted to enter your user name and password. This login page is similar to the login page for the worklist application.

The Excel Workbook includes the task details, such as task ID, payload, and so on. There are buttons corresponding to actions to perform. If you click one of these buttons, the resulting action invokes the BPEL process.

Figure 31-2 provides an overview.

Figure 31-2 Attaching Excel Workbooks to E-mail Notifications



1.

Oracle Business Activity Monitoring

This part describes Oracle Business Activity Monitoring.

This part contains the following chapters:

- [Chapter 32, "Creating Data Objects"](#)
- [Chapter 33, "Using External Data Sources"](#)
- [Chapter 34, "Creating Alerts"](#)
- [Chapter 35, "Using the Oracle BAM Data Control"](#)
- [Chapter 36, "Creating Enterprise Message Sources"](#)
- [Chapter 37, "Using ICommand"](#)

Creating 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 32.1, "Defining Data Objects"](#)
- [Section 32.2, "Adding Permissions on Data Objects"](#)
- [Section 32.3, "Viewing Existing Data Objects"](#)
- [Section 32.4, "Using Data Object Folders"](#)
- [Section 32.5, "Adding Security Filters"](#)
- [Section 32.6, "Adding Dimensions"](#)
- [Section 32.7, "Renaming and Moving Data Objects"](#)
- [Section 32.8, "Adding Indexes"](#)
- [Section 32.9, "Clearing Data Objects"](#)
- [Section 32.10, "Deleting Data Objects"](#)
- [Section 32.11, "System Data Objects"](#)

32.1 Defining Data Objects

This section contains the following topics:

- [Section 32.1.1, "Adding Fields"](#)
- [Section 32.1.2, "Adding Lookup Fields"](#)
- [Section 32.1.3, "Adding Calculated Fields"](#)
- [Section 32.1.4, "Adding Time Stamp Fields"](#)

Data objects contain the information that displays in reports created in Active Studio. You can design data objects to be used in certain types of views such as KPIs, gauges, or columnar reports.

The data objects you define are based on the types of data available from enterprise message sources. You must define fields in the data object. The data object contains no data when you create it. You must load or stream data into data objects using Plans.

When a data object is viewed while creating a report in Active Studio, the data object fields are displayed in alphabetical order regardless of the order that fields were added to the data object.

WARNING: Do not read or manipulate data directly in the database. All access to data must be done using Architect or the ADC API.

To define a data object:

1. Select **Data Objects** from the 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 where the data object will be stored. Click **Browse** to use the **Select a Folder** dialog.
5. Optionally, enter a description of the data object.
6. If this data object will be using an **External Data Source** select the checkbox and configure the following:
 - Select an **External Data Source** from the list. External Data Sources are configured on the External Data Sources screen. See [Chapter 33, "Using External Data Sources"](#) for more information.
 - Enter the **External Table Name**.
7. Add fields to the data object using the **Add a field** or **Add one or more lookup fields** options.

See [Section 32.1.1, "Adding Fields"](#) and [Section 32.1.2, "Adding Lookup Fields"](#) for more information.
8. Click **Create Data Object** when you are finished adding fields or lookup fields.

32.1.1 Adding Fields

To add fields to a data object:

1. In a data object you are creating or editing, click **Add a field**.
2. Specify the field name, data type, maximum size (scale for decimal fields), whether or not it is nullable, whether or not it is public, and tip text.

If you are adding a field in a data object based on an External Data Source you must also supply the **External field name**.

The data types include:

- **String**. Text fields containing a sequence of characters.
- **Integer**. Numeric fields containing whole numbers from -2,147,483,648 to 2,147,483,648.
- **Float**. Double-precision floating point numbers.
- **Decimal**. Integers including decimal points with scale number defined, containing up to 17 digits. The number is stored as a string which uses a character for each digit in the value.
- **DateTime**. Dates and times combined as a real number.
- **Boolean**. Boolean fields with true or false values.
- **Auto-incrementing integer**. Automatically incremented integer field.

- **Timestamp.** Date time stamp generated to milliseconds. See [Section 32.1.4, "Adding Time Stamp Fields"](#) for more information.
- **Calculated.** Calculated field generated by an expression and saved as another data type. See [Section 32.1.3, "Adding Calculated Fields"](#) for more information.

Keep adding fields using **Add a field** and **Add one or more lookup fields** until all the required fields are listed. Click **Remove** to remove a field in the data object.

3. Click **Save changes**.

32.1.2 Adding Lookup Fields

You can add lookup fields to a data object. This performs lookups on key fields in a specified data object to return fields to the current data object. You can match multiple fields and return multiple lookup fields.

To add a lookup field 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 displays.
2. Select the data object to use for the lookup.
3. Select the lookup fields from the data object. You can select one or more fields by holding down the Shift or Control key when selecting. Selecting multiple fields will create multiple lookup fields in the data object. These are the fields you want to return.
4. Select the field to match from the lookup data object.
5. Select the field to match from the current data object. You must have already created other fields in this data object so that you have a field to select.
6. Click **Add**.

The matched field 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 fields. This is also known as a composite key.
8. Click **OK** to save your changes and close the dialog.

The new lookup fields are added to the data object. Click **Modify Lookup Field** to make changes to a lookup field. Multiple selection of return fields 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

32.1.3 Adding Calculated Fields

When creating calculated fields in a data object you can use operators and expression functions, combined with field names, to produce a new field.

[Table 32–1](#) Describes the operators you can use to build calculated fields.

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 field.

Table 32–1 Operators Used in Calculated Fields

Operator	Function
+ (plus sign)	Add
- (minus sign)	Subtract
* (asterisk)	Multiply
/ (forward slash)	Divide
% (percent sign)	Modulus
() (parentheses)	Parentheses

WARNING: If you enter a calculated field with incorrect syntax in a data object, you could lose the data object definition.

32.1.4 Adding Time Stamp Fields

You can create a date time stamp field 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 ADC so that the time stamp data can be created.

32.2 Adding 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.

To add permissions a data object:

1. Select **Data Objects** from the Architect function list.

2. Select the data object.
The general information for the data object displays in the right frame.
3. Click **Permissions**.
4. Click **Edit Permissions**.
Alternatively you can copy permissions from another data object. See [Section 32.2.1, "Copying 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 displays.
6. You can choose to display the following by clicking the radio buttons:
 - 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.
8. After indicating the permissions with selected checkboxes, click **Save changes**.
A message displays to confirm that your changes are saved.
9. Click **Continue** to display the actions for the data object.

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 already exist as a domain group.
3. Click **OK**.
The group is added to the list.

32.2.1 Copying 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 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 Architect function list.
2. Click the data object to add a security filter to.
The general information for the data object displays in the right frame.
3. Click **Permissions**.

4. Click **Copy from**.
The Choose Data Object dialog displays.
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**.

32.3 Viewing Existing Data Objects

This section describes how to view information about data objects. It contains the following topics:

- [Section 32.3.1, "Viewing Data Object General Information"](#)
- [Section 32.3.2, "Viewing Data Object Layouts"](#)
- [Section 32.3.3, "Viewing Data Object Contents"](#)

32.3.1 Viewing Data Object General Information

The general information of a data object displays the owner, when it was created, when it was last modified, and a row count.

To view the general information of a data object:

- Click the data object in the list.

If you are already viewing the layout or contents of a data object, click *General*.

The general information displays 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.** Numbers of rows of data in the data object.
- **Location.**
- **Type.**
- **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.

32.3.2 Viewing Data Object Layouts

The layout describes the fields in a data object. The fields are described by name, field 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 displays in the right frame.
3. Click **Layout**.

The layout information displays in the right frame. It contains the following information:

- **Field name.** Name of the field
- **Field ID.** Generated by the system
- **External name.** External field name from the External Data Source (only appears in data objects based on External Data Sources)
- **Field type.** Data type of the field
- **Max length.** Maximum number of characters allowed in field value
- **Scale.** Number of digits on the right side of the decimal point
- **Nullable.** Whether or not the data type can contain null values
- **Public.** This setting determines whether or not the field will be available in Active Studio to use in a report. If the box is unchecked, the field will not appear in Active Studio. This is useful for including fields for calculations in data objects that should not appear in reports.
- **Lookup.** Displays specifics of a lookup field
- **Calculated.** Displays the expression of a calculated field
- **Tip Text.** Helpful information about the field

32.3.3 Viewing 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 displays in the right frame.

2. Click **Contents**.

The first 50 rows of the data object display in the right frame.

3. Click **Next**, **Previous**, **First**, and **Last** to navigate to other rows of data displayed 50 at a time.

Rows are listed with a Row ID column. Displaying only Row ID provides faster paging for large data objects. Row IDs are assigned once 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.

32.4 Using Data Object Folders

This section contains the following topics:

- [Section 32.4.1, "Creating Folders"](#)
- [Section 32.4.2, "Working with Folders"](#)
- [Section 32.4.3, "Setting Folder Permissions"](#)
- [Section 32.4.4, "Moving Folders"](#)
- [Section 32.4.5, "Renaming Folders"](#)
- [Section 32.4.6, "Deleting 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.

32.4.1 Creating 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 Architect function list.

The current data object folders display in a tree hierarchy.

2. Click **Create subfolder**.

A field for naming the new folder displays.

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 displays confirming that the new folder was created.

4. Click **Continue** to view the folder.

32.4.2 Working with 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 32.11, "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.

32.4.3 Setting 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 displays.
5. You can choose to display the following by selecting one of the radio buttons:
 - 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.
7. After indicating the permissions with selected checkboxes, click **Save changes**. A message displays 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 already exist as a domain group.
3. Click **OK**.

The group is added to the list.

32.4.4 Moving 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.
5. Click **Move folder**.
The folder is moved.
6. Click **Continue**.

32.4.5 Renaming 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**.

32.4.6 Deleting 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 displays to confirm that you want to delete the folder and all of its contents.
3. Click **OK**.
The folder is deleted.
4. Click **Continue**.

32.5 Adding 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.

To add a security filter to a data object:

1. Select **Data Objects** from the Architect function list.

2. Select the data object to add a security filter to.

The general information for the data object displays in the right frame.

3. Select **Security Filters**.

If the data object includes security filters, the filter name and a plus sign displays so that 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 already 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.

Example 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.

32.5.1 Copying 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 Architect function list.
2. Select the data object to add a security filter to.

The general information for the data object displays in the right frame.

3. Select **Security Filters**.

If the data object includes security filters, the filter name and a plus sign displays so that you can expand and view the information.

4. Click **Copy from**.

The Choose Data Object dialog displays.

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**.

32.6 Adding Dimensions

In Architect, you can add dimensions to data objects to define drill paths for charts in Active Studio. Dimensions contain fields 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 within a specific value.

Hierarchies are an attribute of a dimension in a data object. Multiple dimensions can be created in each data object. Each field 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 fields to save as a dimension. Then you organize the fields into a hierarchy.

An example dimension and hierarchy:

Dimension	Hierarchy
Sales	Category
	Brand
	Description

To add a dimension and hierarchy:

1. Select **Data Objects** from the Architect function list.
2. Select the data object to add a dimension to.

The general information for the data object displays 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.
7. Select the field names that you want to include in the dimension. An example is Sales, Category, Brand, and Description.

The field 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 fields.

10. Click **Add new hierarchy**.
11. Enter a hierarchy name.
12. Enter a description for the hierarchy.
13. Select the field names that you want 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 fields is the order that they are listed in the Hierarchy Field list. Arrange the more general grouping field at the top of the Hierarchy Fields list and the most granular field 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.

32.6.1 Time Dimensions

If your dimension contains a time date data type field, 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 field, add a hierarchy.
2. Select the time date data type field. If you are editing existing time levels, click **Edit Time Levels**.

The Time Levels Definition dialog 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.

32.7 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.

To rename a data object:

1. Select **Data Objects** from the Architect function list.
2. Select the data object to rename.

The general information for the data object displays 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**.

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 displays in the right frame.

3. Select **Rename/Move**.
4. Click **Browse** to enter the new location for the data object.
5. Click **Save changes**.

32.8 Adding 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 specific value in a column. If the data object has an index for the fields 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.

You can add indexes to data objects by selecting fields to be indexed as you are creating a data object. You cannot add indexes after loading the data object unless you edit and clear the data object.

To add an index:

1. Select **Data Objects** from the Architect function list.
2. Click the data object to add an index to.
3. Select **Indexes**.

4. Click **Add Index**.

The Add Index dialog opens.

5. Enter a **Name** and **Description** for the index

6. Add as many fields as needed to create an index for the table.

Click a field in the list on the right to remove the field from the index.

7. Click **OK**.

The index is added and is named after the fields it contains. You can create more than one index. To remove an index you created, click **Remove Index** next to the Index name.

32.9 Clearing Data Objects

You might want to clear a data object before loading it. Clearing a data object removes the current contents without deleting the data object from the ADC.

To clear a data object:

1. Select **Data Objects** from the Architect function list.

2. Select the data object to clear.

The general information for the data object displays in the right frame.

3. Click **Clear**.

32.10 Deleting Data Objects

When deleting data objects, make sure that no Plans are accessing the data objects. You should also edit or delete reports and alerts referring to the data object that you want to delete.

To delete a data object:

1. Select **Data Objects** from the Architect function list.

2. Click the data object to delete.

The general information for the data object displays in the right frame.

3. Click **Delete**.

32.11 System Data Objects

The System data objects folder contains data objects used to run Oracle Business Activity Management. 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*.

Using External Data Sources

This chapter contains the information needed to create and manage External Data Sources.

This chapter contains the following topics:

- [Section 33.1, "Introducing External Data Sources"](#)
- [Section 33.2, "Listing External Data Sources"](#)
- [Section 33.3, "Defining External Data Sources"](#)
- [Section 33.4, "Editing External Data Sources"](#)
- [Section 33.5, "Deleting External Data Sources"](#)
- [Section 33.6, "External Data Source Example"](#)

33.1 Introducing External Data Sources

An external data source is a connection to an external database. External data sources usually contain data that does not change very much or data that is too large to bring into the Active Data Cache.

External data source configurations can be exported and imported using ICommand, but you cannot import or edit the contents using ICommand, Enterprise Link, or Architect.

Passwords are entered in clear text. You cannot use DSNs (data source names).

33.2 Listing External Data Sources

To view the existing external data sources:

- Select **External Data Sources** from the Architect function list.

Figure 33–1 Architect Function List



33.3 Defining External Data Sources

To define an external data source:

1. Select **External Data Sources** from the Architect function list.
2. Click **Create**.
3. Enter a name and a description for the external data source.
4. Enter **Driver**, for example, Microsoft ODBC for Oracle.
5. Enter database user credentials in the **Login** and **Password** fields.
6. Enter **Connection string/URL**.

For a complete example of defining an External Data Source against an Oracle database, see [Section 33.6, "External Data Source Example."](#)

33.4 Editing External Data Sources

To edit an external data source:

1. Select **External Data Sources** from the Architect function list.
2. Select the external data source to edit.
The external data source properties display.
3. Select **Edit**.
4. Make the changes and click **Save**.

33.5 Deleting External Data Sources

To delete an external data source:

1. Select **External Data Sources** from the Architect function list.
2. Select the external data source to delete.
The data source properties display.
3. Select **Delete**.
4. Click **OK** to confirm that you want to delete the data source.
The data source is deleted.

33.6 External Data Source Example

This example uses the sample scott/tiger user account and the EMP table in the Oracle database. You may need to unlock the scott/tiger account before proceeding with this example.

Step 1: Create an External Data Source

1. Select **External Data Sources** from the Architect function list.
2. Click **Create**.
3. Enter myDataSource in the **External Data Source Name** field.
4. Enter My Example External Data Source in the **Description** field.
5. Enter Microsoft ODBC for Oracle in the **Driver** field.

6. Enter `scott` in the **Login** field and `tiger` in the **Password** field.

This sample account comes with your Oracle database installation. If you do not have this sample account you can create a new account and use it for this example.

7. Enter `server=net_service_name` in the **Connection string/URL**.

This entry needs to be a Net Service Name defined in your `tnsnames.ora` file.

8. Click **Save**.
9. Click **Continue**.

The External Data Source information is displayed on the screen.

Step 2: Create a Data Object using the External Data Source

1. Select **Data Objects** from the Architect function list.

2. Click **Create Data Object**.

3. Enter `Employees` in the **Name for new Data Object** field.

4. Leave the slash (/) in the **Location for new Data Object** field.

The data object will appear in the top level **Data Objects** folder.

5. Leave the **Tip text** field blank.

6. Enter `Oracle Database Sample EMP Table` in the **Description** field.

7. Select the **External Data Source** checkbox.

8. Select `myDataSource` from the **External Data Source** list.

9. Enter `emp` in the **External Table Name** field.

10. Add the following fields to the data object:

Table 33–1 *Fields in Employees Data Object*

Field	External Field Name	Field Type
<code>ename</code>	<code>ename</code>	String
<code>empno</code>	<code>empno</code>	Integer
<code>job</code>	<code>job</code>	String
<code>mgr</code>	<code>mgr</code>	Integer
<code>hiredate</code>	<code>hiredate</code>	DateTime
<code>sal</code>	<code>sal</code>	Decimal
<code>comm</code>	<code>comm</code>	Decimal
<code>deptno</code>	<code>deptno</code>	Integer

Keep default settings for field attributes not specified in the table.

11. Click **Create Data Object**.
12. Click **Continue**.
13. Click **Contents** to view the contents of the data object

The data in the Employees data object should match the data in the Oracle database sample EMP table.

This chapter describes how to use Alerts.

This chapter contains the following topics:

- [Section 34.1, "Introducing Alerts"](#)
- [Section 34.2, "Creating Alert Rules"](#)
- [Section 34.3, "Using Alert Rule Options"](#)
- [Section 34.4, "Creating Alert Rules From Templates"](#)
- [Section 34.5, "Creating Alert Rules With Messages"](#)
- [Section 34.6, "Creating Complex Alerts"](#)
- [Section 34.7, "Modifying Rules for Alerts"](#)
- [Section 34.10, "Activating Alerts"](#)
- [Section 34.11, "Launching Alerts by URL"](#)
- [Section 34.12, "Deleting Alerts"](#)
- [Section 34.13, "Parameterized Alerts"](#)

34.1 Introducing 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 Architect and Active Studio web applications.

34.2 Creating Alert Rules

A rule specifies the events and conditions under which an alert will fire.

To create a rule:

1. Select **Alerts** in the Architect function list.

In Active Studio, select the **Alerts** tab.

2. Click **Create A New Alert**.

The Rule Creation and Edit dialog displays.

3. Click **Create A Rule**.
4. Enter a name for the rule.
5. Select an event that will launch the alert.
See [Section 34.3.1, "Events"](#) for descriptions of each event.
6. Click **Next**.
7. Select one or more conditions, if needed.
See [Section 34.3.2, "Conditions"](#) for descriptions of each condition.
8. Select one or more actions. See [Section 34.3.3, "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 that displays. 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 navigate between the events page and the page containing actions and conditions, and make changes to those parts of the rule expression you have already 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 34.3.4, "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.

34.3 Using Alert Rule Options

The following sections describe the options for creating alert rules:

- [Section 34.3.1, "Events"](#)
- [Section 34.3.2, "Conditions"](#)
- [Section 34.3.3, "Actions"](#)
- [Section 34.3.4, "Frequency Constraint"](#)

34.3.1 Events

Events launch the rule and trigger the action. Each rule contains only one event.

In a specific amount of time

Select a time interval in seconds, minutes, or hours.

At a specific time today

Select a time.

On a certain day at a specific time

Select a specific date and a time.

Every interval between two times

Select a time interval and two times.

Every date interval starting on certain date at a specific time

Select a date interval such as day, week, month, or year, and a specific date and time.

When a report changes

Select a report to monitor.

When a data field changes in data object

Select a data object and a data field to monitor.

When a data field in a report meets specified conditions

Select a report, the data object, and create a filter on the field to monitor.

When a data field in a data object meets specified conditions

Select a data object and create a filter on the field to monitor.

When this rule is launched

No options to select. When this rule is launched is the event to create dependencies between rules.

34.3.2 Conditions

Conditions are optional settings for constraining the period of time in which the alert is fired. You can select any number of conditions.

If it is between two times

Select two times.

If It is between two days

Select two dates.

If it is a particular day of the week

Select a day of the week.

34.3.3 Actions

Actions are the results of a launched alert. You can select any number of actions.

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.

Send a message via email

Create a message to send and select a recipient.

Send a report via email and escalate to another user after a specific amount of time

Create a message and send to a recipient. Select a secondary recipient to receive the message if the first recipient does not respond within the specified time period.

Send a parameterized message

You can use this option to send reports to other users under the conditions specified. 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**. See [Section 34.13, "Parameterized Alerts"](#) for more information.

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 34.6, "Creating Complex Alerts."](#)

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 34.6, "Creating Complex Alerts"](#)

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.

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. If it is a secure web service check the box and enter the required credentials.
2. Click **Display Services** to display the available services of the URL entered in the field.
3. Click **Map Parameters**. 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.
4. Click **OK** to close the Alert Web Service - Parameter Mapping dialog and the Alert Web Service Configuration dialog.

Run an ODI Scenario

Select the ODI Scenario to run under the specified conditions.

34.3.4 Frequency Constraint

Frequency Constraint can be edited only if it is appropriate for the event selected. otherwise it will be disabled. It can be set to a value of time which could be in seconds, minutes, or hours.

Rules have a five second frequency constraint by default. This limits the number of times the rule will launch in a given time period. With real-time data, transactions can occur every millisecond, so alerting frequency must be controlled. If the rule is

satisfied many times within five seconds, users would not want alerting more than once in five seconds.

34.4 Creating Alert Rules From Templates

Alert rule templates are a convenient preselected group of events and conditions based on some common use cases.

To create an alert rule from a template:

1. Click **Create A New Alert**.
The Create Alert Rule dialog displays.
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 that displays. 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.

34.5 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.

To create an alert rule that includes a message:

1. Start building an alert rule.
2. Select one of the following actions:
 - **Send a message via email**
 - **Send a message via the recipient's alert delivery settings**
3. Click **create message** in the rule expression.
The Alert Message dialog displays.
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. The special fields listed change when reports are selected on the right side of the dialog.

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**.

34.6 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.

To create dependent rules:

1. Create a rule that includes the event **When this rule is launched**. There is no value to specify 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 displays.

4. Select a dependent rule. Only rules that include the **When this rule is launched** event display 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.

34.7 Modifying Rules for Alerts

You modify alert rules from the Alerts tab.

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 displays.
3. Make changes to the alert and click **OK**.

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.

34.8 Viewing Alert History

This functionality is only available in Active Studio.

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 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.

34.9 Clearing Alert History

This functionality is only available in Active Studio.

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 displays to confirm that you want to clear alert history.
2. Click **OK**.

The alert history list is deleted. New alerts launched after clearing will appear in the alert history list.

34.10 Activating 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 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.

Checking the **Activate** checkbox does not cause an alert to launch, it only enables the rule so that if the specified event occurs, the alert will launch.

An exclamation mark on the alert icon indicates it has launched and will not be valid again or because items that it references are missing and it cannot launch.

34.11 Launching Alerts by URL

You can use the alerts web service to manually launch alerts. For more information, refer to:

`http://<host>:<http_port>/oraclebam/services/manualrulefire.asmx?op=FireRuleByName`

You define the rule name using the format:

`DOMAIN\username.alertname`

Note: 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 will not work after a product upgrade. The alert will require editing or the report shortcut will need to be copied again.

34.12 Deleting Alerts

To delete an alert:

1. Select the alert to delete.
2. Click **Delete** in the Alert Actions list.

A dialog displays to confirm that you want to delete the alert.

3. Click **OK**.

The alert is deleted.

34.13 Parameterized Alerts

When creating a parameterized alert, you must populate the **set parameters** section. In this section, populate the **User**, **Delivery**, and **Report** fields with either predefined values or dynamically from a Data Object field.

- **User field**

If you populate this field with predefined values, the value that appears must follow a format such as `MY-DOMAIN\myhost-pc`. If you populate this field from a Data Object field, the value also follows a format such as `MY-DOMAIN\myhost-pc`.

- **Delivery field**

If you populate this field with predefined values, the value that appears in this field is `Email` (or something similar). If you populate this field from a Data Object field, the value must be `smtp`.

- **Report field**

If you populate this field with predefined values, the value that appears in this field is `Emp_Report` (or something similar). If you populate this field from a Data Object field, the value must be the report ID of that report, and not the name. To get the report ID, click the report (for example, `Emp_Report`) and click the **Copy Shortcut** link. A window displays with a link such as:

<http://SERVER1/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.

Using the Oracle BAM Data Control

The Oracle BAM data control is a binding component in the Oracle ADF Model. This chapter provides information about creating and using the Oracle BAM data control.

For more comprehensive information about using Oracle ADF Model data binding, refer to *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

This chapter contains the following topics:

- [Section 35.1, "Introduction to the Oracle BAM Data Control"](#)
- [Section 35.2, "Creating Oracle BAM Server Connections"](#)
- [Section 35.3, "Creating Projects That Can Use Oracle BAM Data Controls"](#)
- [Section 35.4, "Exposing Oracle BAM with Oracle ADF Data Controls"](#)
- [Section 35.5, "Creating Oracle BAM Data Control Queries"](#)
- [Section 35.6, "Using Oracle BAM Data Controls in ADF Pages"](#)

35.1 Introduction to the Oracle BAM Data Control

The Oracle BAM data control allows ADF developers to build applications with a dynamic user interface that changes based on real-time business events. The Oracle BAM data control is used to bind data from Oracle BAM data objects to databound UI components in an ADF page.

The Oracle BAM data control abstracts a query on Oracle BAM data objects using standard metadata interfaces to describe the Oracle BAM data collections. Using JDeveloper, you can view that information as icons which you can drag and drop onto a page. Using those icons, you can create databound UI components (for JSF JSP pages) by dragging and dropping them from the Data Controls panel onto the visual editor for a page. JDeveloper automatically creates the metadata that describes the bindings from the page to the Oracle BAM data objects. At runtime, the ADF Model layer reads the metadata information from appropriate XML files for both the data controls and bindings and implements the two-way connection between your user interface and your Oracle BAM data objects.

35.2 Creating Oracle BAM Server Connections

You must create a connection to Oracle BAM to browse the available data objects in JDeveloper. This connection information is automatically used during deployment.

Note: You can create connections to Oracle BAM in the Oracle JDeveloper Resource Palette as well as the Application Resources panel of a specific application. It is recommended that you create these connections in the Application Resources pane rather than the Resource Palette.

To create an Oracle BAM connection:

1. Select **New** from the **File** main menu in Oracle JDeveloper.
The New Gallery dialog opens.
2. Choose **Connections** from the **General** category.
3. Select **BAM Connection** in the **Items** list, and click **OK**.
The BAM Connection wizard opens.
4. Provide a name for the connection. Leave the **Create Connection In** selection as **Application Resources**.
5. Click **Next**.
6. Enter the following connection information about the Oracle BAM instance.

Field	Description
BAM Web Host	Enter the name of the host on which the BAM report server and Web server are installed. In most cases, the BAM Web 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 (typically <code>bamadmin</code>).
Password	Enter the password of the user name.
HTTP Port	Enter the port number or accept the default value of 8888 . This is the HTTP port for the BAM Web host.
RMI Port	Enter the port number or accept the default value of 9085 . The RMI port is for the BAM report cache, which is part of the Oracle BAM Server.
Use HTTPS	Select this check box if you want to use secure HTTP (HTTPS) to connect to the Oracle BAM Server during design time. Otherwise, HTTP is used.

7. Click **Next**.
8. Test the connection by clicking **Test Connection**. If the connection was successful, the following message appears:
`Passed.`
9. Click **Finish**.

35.2.1 How to Modify Oracle BAM Data Control Connections to Oracle BAM Servers

Each Oracle BAM data control has an associated Oracle BAM connection. When a connection has changed name or has been removed from the application resources, you will get an error when you attempt to use any data controls that are associated with the connection. You can do one of the following to resolve the lost connection:

- Create a new Oracle BAM connection with the same name as the connection that is referred to by the data control. See [Section 35.2, "Creating Oracle BAM Server Connections"](#) for more information.
- Update the current project's DataControls.dcx file with the name of a new or existing Oracle BAM connection. See [Section 35.2.1, "How to Modify Oracle BAM Data Control Connections to Oracle BAM Servers"](#) for more information.

35.2.1.1 How to Associate a BAM Data Control with a New Oracle BAM Connection

To change the Oracle BAM connection associated with a particular data control you must edit the DataControls.dcx file in the current project. Change the `connection` attribute of the `BAMDataControl` element with the name of the desired Oracle BAM connection.

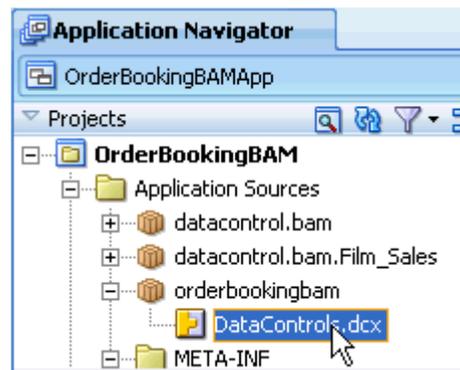
To modify the Oracle BAM connection in an Oracle BAM data control:

1. Optionally, create a new Oracle BAM connection in the application.

If you do not already have a BAM connection in the Application Resources that you want to use for this data control, create a new one. See [Section 35.2, "Creating Oracle BAM Server Connections"](#) for more information.

2. Locate the DataControls.dcx file in the project, and open it for editing.

The DataControls.dcx file is located in the Application Sources directory under the node named for the project.



Each project in a JBuilder application has a DataControl.dcx file associated with it. Each DataControls.dcx file may have one or more data control definitions. If the current project does not contain the definition for the data control you wish to modify, look through the other projects in the current application to locate it.

3. In the Source view, locate the appropriate data control definition, and locate the `BAMDataControl` element within it.

In the source view find the `AdapterDataControl` block with the `id` that matches the display name of your data control.

```

<AdapterDataControl id="Film_Sales"
FactoryClass="oracle.tip.tools.ide.bam.dc.dt.ad
ImplDef="oracle.tip.tools.ide.bam.dc.dt.adapter.
SupportsTransactions="false"
SupportsSortCollection="false" SupportsResetStat
SupportsRangeSize="false" SupportsFindMode="fals
SupportsUpdates="false"
Definition="datacontrol.bam.Film_Sales"
BeanClass="datacontrol.bam.Film_Sales"
xmlns="http://xmlns.oracle.com/adfm/datacontrol'

<Source>
<BAMDataControl xmlns="http://xmlns.oracle.com/bam/datacontrol"
connection="BAMServerConnection1">

```

4. Change the `connection` attribute to the name of the new Oracle BAM connection.
5. Save and close the `DataControls.dcx` file.

35.3 Creating Projects That Can Use Oracle BAM Data Controls

A limited set of ADF Faces components support active data, therefore a limited set of ADF Faces components can make use of the main functionality of an Oracle BAM data control. These technologies must be part of the project; either they are added when the application is created on an ADF-ready template, or the appropriate tag libraries can be added to the project later.

Note that an Oracle BAM data control can still be used by view components that do not support active data.

The Oracle BAM data control requires that the project contain the ADF Faces and ADF Pages Flow technologies. The Fusion Web Application (ADF) template in JDeveloper contains these technologies.

However, if you are planning to add an Oracle BAM data control to a project and application that was not created using an ADF template, you can add the ADF Data Visualization and ADF Faces Components tag libraries to the project.

To add ADF Data Visualization and ADF Faces Components tag libraries to a project:

1. In the Application Navigator, right-click the project node and select **Project Properties**.
2. In the Project Properties dialog, select **JSP Tag Libraries**, and click **Add**.
3. Select both **ADF Data Visualization** and **ADF Faces Components** tag libraries, and click **OK**.
4. Click **OK** to close the Project Properties dialog.

35.4 Exposing Oracle BAM with Oracle ADF Data Controls

Once you have created your Oracle BAM data objects and established a connection to an Oracle BAM server from JDeveloper, you can use JDeveloper to create data controls that provide the information needed to declaratively bind UI components to those

data objects. Data controls consist of a number of XML metadata files that define the capabilities of the service that the bindings can work with at runtime.

See [Chapter 32, "Creating Data Objects"](#) for information about creating Oracle BAM data objects. For information about creating a connection to your Oracle BAM instance, see [Section 35.2, "Creating Oracle BAM Server Connections."](#)

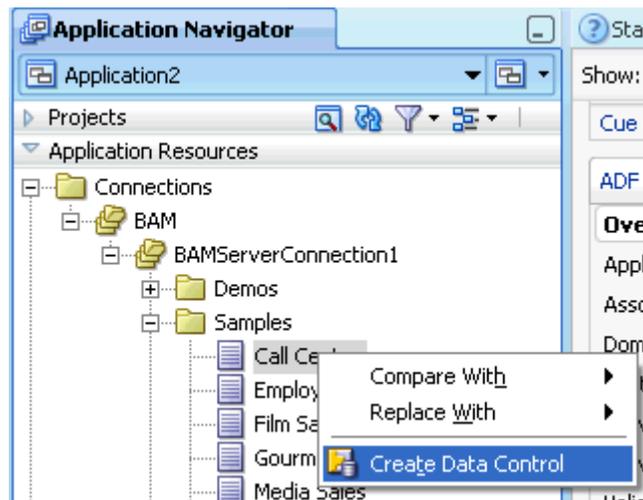
35.4.1 How to Create Oracle BAM Data Controls

You create Oracle BAM data controls from within the Application Navigator of JDeveloper.

To create a data control:

1. In the Application Navigator, Application Resources panel, right-click the Oracle BAM data object for which you want to create a data control.
2. From the context menu, choose **Create Data Control**.

Figure 35–1 Oracle BAM Data Object Context Menu



3. Complete the BAM Data Control wizard to create the data control query.
See [Section 35.5, "Creating Oracle BAM Data Control Queries"](#) for more information.

35.4.2 What Happens in Your Project When You Create an Oracle BAM Data Control

When you create a data control based on an Oracle BAM data object, the data control contains a representation of a query on all of the selected fields that is constructed based on the groupings, aggregates, filters, parameters, and additional calculated fields that you configure using the BAM Data Control wizard in JDeveloper.

For the data control to work directly with the service and the bindings, JDeveloper creates the following metadata XML files:

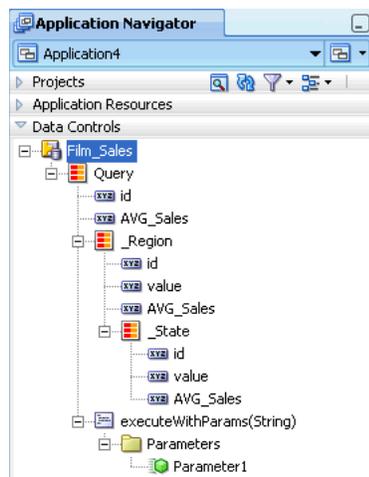
- Data control definition file (`DataControls.dcx`)
- Structure definition files for every structured object that this service exposes
- Design time XML files

JDeveloper also adds the icons to the Data Controls panel that you can use to create data bound UI components.

35.4.2.1 How an Oracle BAM Data Control Appears in the Data Controls Panel

The Data Controls panel lists all the data controls that have been created for the application's business services and exposes all the queries that are available for binding to UI components. The panel is a direct representation of the metadata XML files created when creating a data control. By editing the data control, you can change the elements displayed in the panel.

Figure 35–2 Data Controls Panel in JDeveloper



35.5 Creating Oracle BAM Data Control Queries

You can design a databound user interface by dragging an item from the Data Controls panel and dropping it on a page as a specific UI component. When you use Oracle BAM data controls to create a UI component, JDeveloper automatically creates the various code and objects needed to bind the component to the data control you selected.

You can create an Oracle BAM data control query using the Oracle BAM Data Control wizard. The wizard lets you choose between creating a flat query or a group query.

The following sections explain how to use each page in the wizard to create your query:

- [Section 35.5.1, "Choosing a Query Type"](#)
- [Section 35.5.2, "How to Create Parameters"](#)
- [Section 35.5.3, "How to Create Calculated Fields"](#)
- [Section 35.5.4, "How to Select, Organize, and Sort Fields"](#)
- [Section 35.5.5, "How to Create Filters"](#)
- [Section 35.5.6, "How to Select and Organize Groups"](#)
- [Section 35.5.7, "How to Create Aggregates"](#)

35.5.1 Choosing a Query Type

On the Name page of the Oracle BAM Data Control wizard you can choose to create either a flat query or a group query.

Create a flat query for the Oracle BAM Data Control when you want to show the data in a flat table or list.

Create a group query for the Oracle BAM Data Control when you want to create groups and aggregates of data to display in trees or charts.

35.5.2 How to Create Parameters

On the Parameters page of the Oracle BAM Data Control wizard you can create parameters that are used to pass values to filters on the Filters page of the wizard.

To create parameters:

1. Click **Add** to add a parameter.
2. To rename the parameter enter the text in the **Name** field.
3. Select the data type from the **Type** list.

Table 35–1 Oracle BAM and Java Type Mapping

Java Type	Oracle BAM Type
java.lang.Integer	Integer
java.lang.String	String
java.util.Date	DateTime, Timestamp
java.lang.Boolean	Boolean
java.lang.Long	Decimal
java.lang.Double	Float

4. To provide a default value for the parameter when loading the data control query, select **Enable Default Value** and choose a default value.

To enter a default value for the parameter, select one of the available defaults, or enter a value in the field.

- **ALL**
Returns rows containing all values.
- **NULL**
Returns rows containing null values
- **BLANK**
Returns rows containing blank string values.

35.5.3 How to Create Calculated Fields

Calculated fields allow you to create new columns based on data derived from existing fields without updating the physical data object. Use the Oracle BAM Data Control wizard Calculated Fields page to create them.

To create calculated fields:

1. Click **Add** to add a calculated field.
The new default field name appears in the list of calculations.
2. Click **Rename** to change the display name of a calculated field.
3. To enter an expression, choose an expression from the expressions list, and click **Insert Expr** and complete the expression.

There are several preformed expressions available. See *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for examples and more information about each expression.

4. To use a data object field in a calculation, select the field from the field list, and click **Insert Field**.

35.5.3.1 Creating Groups in Calculated Fields

You can create groups in the calculations page.

To create groups on calculations:

1. Select a calculation in the calculations list.
2. Click **Group By**.
3. Choose a field to group by, and click **OK**.

35.5.4 How to Select, Organize, and Sort Fields

To deselect all of the fields, uncheck the **ALL** check box, and select individual fields.

The field at the top of the list will appear in the left-most column of the final table in the ADF page. To change the order in which the fields appear, select a field and use the blue arrows to move it up or down the list.

To apply sorting on a field, click the sorting type in the **Sorting** column, and choose a new sorting type from the list.

Note: If you use Active Data, sorting will be preserved on Update, Upsert operations, but not on Insert operations.

35.5.5 How to Create Filters

You can apply filters to both Group Query and Flat Query types of Oracle BAM data controls. Add combinations of entries and headers to create complex filter expressions.

35.5.5.1 How to Create Filter Headers**To create a sub-header under an existing header:**

1. In the Filters page of the Create BAM Data Control wizard, select a header under which to add the sub-header, and click **Add Header**.

You can select the main header at the top of the filter expression to create a sub-header under it.

2. To change the operator (default **ALL**), select the header, and click **Edit**. For the following operator options, data is returned when:
 - **ALL**. All of the included entries are true.

- **NONE.** None of the included entries are true.
 - **AT LEAST ONE.** At least one and maybe more of the included entries are true.
 - **NOT ALL.** Some or none of the included entries are true, but not all of the included entries are true.
3. Select an operator from the **Filters** list, and click **OK**.

35.5.5.2 How to Create Filter Entries

To create a filter entry:

1. In the **Filters** page of the Create BAM Data Control wizard, select a header under which to add the filter entry. For information about creating headers in the filter expression see <creating header xref>.
2. Click **Add Filter Entry**.
The Add Filter Entry dialog opens.
3. Choose a field from the **Field** list.
4. Choose an expression from the **Comparison** list. Choices include:
 - **is equal to** returns rows containing an exact value match.
 - **is not equal to** returns rows containing all values except specified value.
See [Section 35.5.5.3, "Entering Comparison Values"](#) for information on configuring comparison values.
 - **is less than** returns rows containing values less than specified value.
 - **is less than or equal to** returns rows containing values less than or equal to specified value.
 - **is greater than** returns rows containing values greater than specified value.
 - **is greater than or equal to** returns rows containing values greater than or equal to specified value.
 - **is like** returns rows containing values that match a string pattern. Include an underscore (_) as a wildcard for a single character in a string and a percent symbol (%) as a wildcard for one character or more. Wildcard characters can be combined, for example, %mm_00 would return all columns (35mm 200, 35mm 400, 35mm 800). Do not enter any spaces in the expression since spaces are treated as characters in the data match.
 - **is not like** returns rows containing values that do not match a string pattern.
 - **is null** returns rows containing values where the column is null. If you select this comparison, your filter configuration is complete. Click **OK** to create the filter. For numeric data types, nulls are not returned for filters returning values equal to zero. In other words, zeroes are not treated as null values. A null represents missing data in the field.
 - **is not null** returns rows containing values where the column is not null. If you select this comparison, your filter configuration is complete. Click **OK** to create the filter. For numeric data types, nulls are not returned for filters returning values equal to zero. In other words, zeroes are not treated as null values. A null represents missing data in the field.

- **is in list** returns rows containing values included in a list. To build a list, click **Edit**. Type a value in the field and click **Add** to add it to the list. Add as many values as needed. Click **Browse** to choose values currently present in the Data Object. Click **Remove** to remove a value. Click **OK** to close the dialog.
- **is not in list** returns rows containing values not included in the list. To build a list, click **Edit**. Type a value in the field and click **Add** to add it to the list. Add as many values as needed. Click **Browse** to choose values currently present in the Data Object. Click **Remove** to remove a value. Click **OK** to close the dialog.
- **is within a time interval** returns rows containing values that occur within the specified time interval. Configure the time interval using the provided lists. Select a **Type**, enter a multiplier in the field and select a **Unit**.

When filtering on a datetime or timestamp field, you can enable **Active Now** to keep the displayed time interval current as time passes. Configure the **Active Now Interval** to specify how often to refresh the display. See [Section 35.5.5.4, "Using Active Now"](#) for more information.

- **is within the current time period** returns rows containing values that occur within the current specified time unit. Select a **Unit** from the list.

When filtering on a datetime or timestamp field, you can enable **Active Now** to keep the displayed time period current as time passes. See [Section 35.5.5.4, "Using Active Now"](#) for more information.

- **is within a time period** returns rows containing values that occur within the specified time period. Configure the time period using the provided lists. Enter a value in the **Offset** field, select a **Unit**, and select a **Type**.

When filtering on a datetime or timestamp field, you can enable **Active Now** to keep the displayed time period current as time passes. See [Section 35.5.5.4, "Using Active Now"](#) for more information.

5. Click **OK** to add the entry to the filter expression.

35.5.5.3 Entering Comparison Values

For most **Comparison** values you must choose **Value**, **Field**, or **Calculation** from the **Value** list.

35.5.5.3.1 Comparison With a Value If you select **Value**, do one of the following:

- Click **Browse** to see a list of values present in the Data Object. Select a value from the list. Up to 50 values display in the list. The field can be left blank to create a filter on a blank string.

Note: If there are more than 50 values in the field, not all of the values are shown in the **Browse** list. Your Oracle Business Activity Monitoring administrator can configure the number of rows to display in the list. See the *Oracle Business Activity Monitoring Installation Guide* for more information.

- Manually enter a value in the field.

35.5.5.3.2 Comparison With a Field If you select **Field**, select a field from the last list to compare with the field selected in the **Field** list.

35.5.5.3.3 Comparison With a Calculation If you select **Calculation**, enter an expression in the field to compare with the first field.

For example, if you create a list view using the sample Call Center data object and create a filter with the following attributes:

- **Field.** Total
- **Comparison.** is equal to
- **Value.** Calculation
- **Calculation field.** Quantity*2

This filter will yield only those rows where the value in the Total column is equal to twice the value in the Quantity column.

35.5.5.4 Using Active Now

The Active Now feature in data filtering enables you to display in your views a segment of the data that is always within a defined time window. As time passes, the view is updated with the data within the defined time interval in the filter. Older data is removed from the view and newer data is added as time passes.

Active Now is available when you choose one of the following comparison expressions:

- **is within a time interval**
- **is within the current time period**
- **is within a time period**

Active Now behaves differently depending on which comparison expression you choose.

When you choose **is within a time interval**, you can control how often the data is refreshed using the **Active Now Interval** setting.

For example, if you create a filter using **is within a time interval**, **previous** type, **1, Hours** unit, and **Active Now**, set the **Active Now Interval** to 60 seconds, and the current time is 3:25 p.m., data from 2:25 p.m. - 3:25 p.m. is displayed in the view. When the current time changes to 3:26 p.m., data from 2:26 p.m. - 3:26 p.m. is displayed in the view. Every 60 seconds the oldest minute of data is removed from the view and the newest minute is added.

When you choose **is within the current time period** or **is within a time period**, the data is refreshed when the time period changes.

For example, when you create a filter using **is within the current time period**, the **Hours** unit, and **Active Now**, and the current time is 3:25 p.m., only data from 3:00 p.m. - 3:59 p.m. is displayed in the view until the current time is 4:00 p.m. At 4:00 p.m. all the data from 3:00 p.m. - 3:59 p.m. is removed from the view, and data that accumulates during the 4:00 p.m. - 4:59 p.m. time interval is displayed in the view.

35.5.6 How to Select and Organize Groups

To specify a group:

1. In the Groups page of the Create BAM Data Control wizard, select one or more fields in the **Group Fields** list.

To group by numeric fields, first select **Show Numeric Fields** at the bottom of the list.

2. To change the display order in which the groups will be presented in a graph, select a sorting option from the **Sorting** list for any selected field.
3. If a datetime field is selected in the **Fields** list, several options are enabled for configuring Time Groups on the right side of the wizard page.
See [Section 35.5.6.1, "How to Configure Time Groups and Time Series"](#) for more information.

35.5.6.1 How to Configure Time Groups and Time Series

You can create a chart where the grouping (x axis) is based on a datetime field.

To configure time groups:

1. In the Groups page of the Create BAM Data Control wizard, select a single field of type datetime in the **Group Fields** list.

This action enables the Time Groups options on the right side of the wizard page.

2. Select **Continuous Time Series** to display empty groups for time intervals where no data is available.

There may be time gaps where the data object did not have entries. The Continuous Time Series feature adds groups to the result whose values are zero, so that when the results are shown on the graph, the x axis represents a smooth time series.

Continuous Time Series is valid only if you have chosen a single datetime field to group by. Continuous Time Series is not supported if any additional group fields are selected.

3. Select either **Use Time Series** or **Use Time Groups**.
 - **Use Time Series** displays the data from the first datetime data point available in the data object to the last in the configured time interval.
 - **Use Time Groups** displays data grouped into a set number of time intervals. For example, if you select Month from the time unit list, all data from January from all years where data is available will be grouped in one data point on the chart.
4. Select a time unit from the list.

If you selected **Use Time Groups**, the groups are described in the following list.

- **Year** displays groups for all of the years where data is available.
- **Quarter** displays four groups representing the quarters of a year (January-March, April-June, July-September, and October-December).
- **Month** displays twelve groups representing the months of the year.
- **Week** displays 52 groups representing the weeks in a year.
- **Day of Year** displays groups representing the 365 possible days in a year.
- **Day of Month** displays 31 groups representing the possible days of a month.
- **Day of Week** displays seven groups representing the days of the week.
- **Hour** displays 24 groups representing the hours of a day.
- **Minute** displays 60 groups representing the minutes in an hour.
- **Second** displays 60 groups representing the seconds in a minute.

5. Enter a quantity of the time unit to group by. For example, entering a **2** next to the Month time unit will display the groups in two month increments (January and February will be grouped as one data point on the chart).
6. Click **Next** or **Finish**.

35.5.7 How to Create Aggregates

To specify an aggregate on a field:

1. In the Aggregates page of the Create BAM Data Control wizard, select a field in the **Fields** list.

The valid **Summary Functions** for the data type of that field are enabled.

2. Select one or more valid **Summary Functions**.

The expressions appear in the Summary Values list.

35.5.8 How to Modify the Query

To edit the Oracle BAM data control, right click the data control node, and select **Edit Definition**. The Edit BAM Data Control wizard opens and you can jump to any page to edit that part of the query.

35.6 Using Oracle BAM Data Controls in ADF Pages

Oracle BAM data controls can be used a subset of ADF Faces components in a JSF page.

To use the Oracle BAM data control in a JSF page:

1. Set the default browser:
 - a. In the JDeveloper Tools menu, select **Preferences**.
 - b. In the Preferences dialog, select **Web Browser and Proxy**.
 - c. Choose a default browser by entering the path to the browser's executable in the **Browser Command Line** field, enter any applicable proxy information, and click **OK**.
2. Create a JSF page:
 - Right click project, select **New**, select **JSF** under the Web Tier category, and select **JSF JSP** in Items list.
 - Check the option **Create as XML Document (*.jspx)** and uncheck **Render in Mobile Device**.
3. Drag and drop an accessor node from the Data Controls panel to the JSF page editor.
4. Select a data visualization component.

A subset of ADF components support active data. See the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* for more information about binding data controls with data visualization components.
5. Save all, and in the Oracle JDeveloper toolbar, click **Run Project**.

Creating Enterprise Message Sources

This chapter contains the information needed to create and manage Oracle BAM enterprise message sources.

This chapter contains the following topics:

- [Section 36.1, "Introducing Enterprise Message Sources"](#)
- [Section 36.2, "Listing Enterprise Message Sources"](#)
- [Section 36.3, "Defining Enterprise Message Sources"](#)
- [Section 36.4, "Editing Enterprise Message Sources"](#)
- [Section 36.5, "Copying Enterprise Message Sources"](#)
- [Section 36.6, "Deleting Enterprise Message Sources"](#)

36.1 Introducing Enterprise Message Sources

Enterprise message sources 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.

Each enterprise message source 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 enterprise message sources.

The following JMS providers are supported:

- Oracle Enterprise Message Service JMS In-Memory and File-Based resource provider
- Oracle Enterprise Message Service Database resource provider
- Non-Oracle JMS providers
 - WebSphereMQ
 - Tibco
 - SonicMQ

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.)

36.2 Listing Enterprise Message Sources

To view the existing enterprise message sources:

- Select **Enterprise Message Sources** from the Architect function list.

Figure 36–1 Architect Function List



36.3 Defining Enterprise Message Sources

When you define an enterprise message source, 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 enterprise message source, you must be familiar with the third party application providing the messages so that you can specify the message source connection details in Architect.

To define an enterprise message source:

1. Select **Enterprise Message Sources** from the Architect function list.
2. Click **Create**.
3. Using [Table 36–1](#) as a guide, enter the appropriate values in each of the fields.
4. If you are using TextMessage type, configure the appropriate parameters in the XML Formatting sections, using [Table 36–2](#) as a guide.
5. Source Value Formatting.
6. Map fields from the source message to the Oracle BAM data object in the **Source to Data Object Field Mapping** section.

Table 36–1 Enterprise Message Source Configuration Parameters

Parameter	Description
Name	A unique display name that appears in the Enterprise Message Sources list in Architect.

Table 36–1 (Cont.) Enterprise Message Source Configuration Parameters

Parameter	Description
Initial Context Factory	The initial context factory to be used for looking up specified JMS connection factory or destination. For example: oracle.j2ee.rmi.RMIInitialContextFactory
JNDI Service Provider URL	Configuration information for the service provider to use. Used to set javax.naming.Context.PROVIDER_URL property and passed as an argument to initialContext(). An incorrect provider URL is the most common cause of errors. For example: ormi://localhost:23791
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 javax.naming.Context.SECURITY_PRINCIPAL and passed to initialContext().
JNDI Password	The identity of the principal for authenticating the JNDI service caller. Used to set javax.naming.Context.SECURITY_CREDENTIALS and passed to initialContext().
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 36–2 for more information.
Durable Subscriber Name	
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 stringProperty() 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 enterprise message source. 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	
Transaction	

Table 36–1 (Cont.) Enterprise Message Source Configuration Parameters

Parameter	Description
Start when BAM Server starts	
JMS Username (Optional)	You can optionally provide this information when a new JMS connection is created by a connection factory.
JMS Password (Optional)	Used to authenticate a connection to a JMS provider for either application-managed or container-managed authentication.

Table 36–2 Enterprise Message Source 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 36.3.1, "Using Advanced XML Formatting" for more information. XML names can be qualified. If qualified, check 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, check 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.
Column Value	Column values can be provided using either elements or attributes in an XML payload. Specify which column value type will be provided in the payload.

Table 36–3 Enterprise Message Source Source Value Formatting Configuration Parameters

Parameter	Description
Pattern	
Locale	

36.3.1 Using Advanced XML Formatting

The Advanced formatting options allow an enterprise message source to contain a user-supplied XSL Transformation (XSLT) for each formatted field in the message.

Uses for XSL transformations include:

- Handling of hierarchical data. The Data Flow does not handle hierarchical data. The XSL transformation can flatten the received XML into a single record with repeating fields.
- Handling of message queues that contain messages of more than one type in a single queue. The Data Flow requires that all records from the Message Receiver be of the same schema. The enterprise message source output can be defined as a combined superset of the message schemas that will be 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 more than one level 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.
- Handling changes to message formats without affecting Plans. If a complex message-processing Plan exists and the format of the messages, or the data in them, is changed slightly, the XSL transformation could compensate for this change so that changes to the Plan are not required.

To specify an XSL transformation:

1. In an enterprise message source that you are defining or editing, select one of the XML formatting options in the Formatting column.
2. Click **Advanced formatting options**.
The Advanced Formatting dialog displays.
3. Type or paste the XSL code for the transformation for the XML in this field. You might want to write the XSL in another editing tool and then copy and paste the code into this dialog.
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 and will not be displayed if you close and open this dialog.
5. Click **Verify transformation syntax** to check 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.

36.3.2 XSL Processing and Example Code

This section presents an example.

36.3.2.1 XSL Processing in an Example Enterprise Message Source

The Enterprise Message Receiver Transform receives messages from the queue, and converts them to a traditional row and column format to send into the Data Flow.

The records in the Data Flow must be flat records because hierarchical data is not supported. The Message Receiver can generate multiple Data Flow records for each received message, but all records must have the same schema.

In the Enterprise Message Source definition, a formatting specification can be provided for any fields in the message of type String to parse XML inside the string field. Typically for MSMQ, the Body field of the message contains the XML for the message.

The formatting specification indicates how the received XML is to be converted into one or more records. It assumes that the fields of data are contained within a particular single node-type within the XML, either as attributes of that node or as tags directly contained in that node. Although this formatting can handle multiple instances of the node, it cannot handle multiple node types, nor can it handle hierarchies in the XML.

The following example shows how a simple message might look:

```
<order>
```

```

<item>
  <description>Fuel pump</description>
  <price>128.95</price>
  <quantity>1</quantity>
</item>
<item>
  <description>Fuel filter</description>
  <price>12.95</price>
  <quantity>1</quantity>
</item>
</order>

```

The generated records for the preceding message might look like the following:

Description	Price	Quantity
Fuel pump	128.95	1
Fuel filter	12.95	1

36.3.2.2 Handling Complex Messages

To handle complex XML, or do formatting or parsing not supported by the Enterprise Message Receiver, the Enterprise Message Source can contain an XSL-T specification to "preprocess" the received XML before it is passed to the Message Receiver formatting. This transformation can be complex, and can contain any code supported by the Microsoft .NET XSL-T Processor, including scripting. Any field in the received message of type String can have an XSL-T preprocessor. In the case of MSMQ, this is almost always the Body field of the message.

The XSL-T code can be used for any purpose required. For this example, it is used to translate hierarchical message XML into simpler XML that can be handled by the formatting. Typically, this would mean flattening of the hierarchical message into multiple records, where the higher-level field values repeat.

Phoenix Debt Order Messages

The XML in these messages expresses a hierarchy. Each message contains a root PhoenixDebtOrder tag, which contains one or more PhoenixDebtOrderProduct tags, which each contain one or more PhoenixDebtOrderIOI tags:

```

<PhoenixDebtOrder attr1 attr2>
  <PhoenixDebtOrderProduct attr3 attr4>
    <PhoenixDebtOrderIOI attr5 attr6>

```

The values at each level are contained in attributes on the tags.

The purpose of the XSL transformation is to convert the preceding hierarchy into a flat structure containing a node for each PhoenixDebtOrderIOI node, with the values for the parent tags repeated in each instance:

```

<PhoenixDebtOrder>
  <Flat attr1 attr2 attr3 attr4 attr5 attr6>

```

The XSL assumes that the attribute names are different at each level, or if they are not different, that they represent the same information and do not need to be repeated.

The XSL does a generic copy of all of the attributes at each level, and no specific attribute names are referenced. If attributes are added or removed at any level, it is not necessary to change the XSL code, although in such cases, it is necessary to change the formatting in the Enterprise Message Source.

36.3.2.3 Phoenix Debt Order Example

The following text shows a sample Phoenix Debt Order message:

```
<PhoenixDebtOrder ord_id="1847494630" rgn_id="0"
  last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
  iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
  identity_nm="">
<PhoenixDebtOrderProduct ord_id="1847494630" prd_id="1847494611"
  price_basis="spread" ioi_ccy_id="1847483827" canceled_ind="0" inst_alloc_qty=""
  ret_alloc_qty="">
<PhoenixDebtOrderIOI ord_ioi_id="1847558620" ord_id="1847494630"
  ioi_prd_id="1847494611" ioi_size="4770000000" ioi_px="14.123512"/>
<PhoenixDebtOrderIOI ord_ioi_id="1847558621" ord_id="1847494630"
  ioi_prd_id="1847494611" ioi_size="5540000000" ioi_px="15.252500"/>
<PhoenixDebtOrderIOI ord_ioi_id="1847558619" ord_id="1847494630"
  ioi_prd_id="1847494611" ioi_size="3330000000" ioi_px="12.500000"/>
</PhoenixDebtOrderProduct>
<PhoenixDebtOrderProduct ord_id="1847494630" prd_id="1847494612"
  price_basis="spread" ioi_ccy_id="1847483827" canceled_ind="0" inst_alloc_qty=""
  ret_alloc_qty="">
<PhoenixDebtOrderIOI ord_ioi_id="1847558620" ord_id="1847494630"
  ioi_prd_id="1847494612" ioi_size="5880000000" ioi_px="14.124512"/>
<PhoenixDebtOrderIOI ord_ioi_id="1847558621" ord_id="1847494630"
  ioi_prd_id="1847494612" ioi_size="4430000000" ioi_px="12.252500"/>
</PhoenixDebtOrderProduct>
</PhoenixDebtOrder>
```

The following text shows the transformed XML produced for the previous example:

```
<PhoenixDebtOrder>
<Flat ord_id="1847494630" rgn_id="0"
  last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
  iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
  identity_nm="" prd_id="1847494611" price_basis="spread" ioi_ccy_id="1847483827"
  canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558620"
  ioi_prd_id="1847494611" ioi_size="4770000000" ioi_px="14.123512"/>
<Flat ord_id="1847494630" rgn_id="0"
  last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
  iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
  identity_nm="" prd_id="1847494611" price_basis="spread" ioi_ccy_id="1847483827"
  canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558621"
  ioi_prd_id="1847494611" ioi_size="5540000000" ioi_px="15.252500"/>
<Flat ord_id="1847494630" rgn_id="0"
  last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
  iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
  identity_nm="" prd_id="1847494611" price_basis="spread" ioi_ccy_id="1847483827"
  canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558619"
  ioi_prd_id="1847494611" ioi_size="3330000000" ioi_px="12.500000"/>
<Flat ord_id="1847494630" rgn_id="0"
  last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
  iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
  identity_nm="" prd_id="1847494612" price_basis="spread" ioi_ccy_id="1847483827"
  canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558620"
  ioi_prd_id="1847494612" ioi_size="5880000000" ioi_px="14.124512"/>
<Flat ord_id="1847494630" rgn_id="0"
  last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
  iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
  identity_nm="" prd_id="1847494611" price_basis="spread" ioi_ccy_id="1847483827"
  canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558621"
  ioi_prd_id="1847494612" ioi_size="4430000000" ioi_px="12.252500"/>
</PhoenixDebtOrder>
```

36.3.2.4 Sequence Numbers

A single parameter is passed to the XSL transformation from the Message Receiver Transform, called "SequenceNumber". This is an integer that starts at one and increments for each message received. This number can be used to identify the set of records generated for a single original message, or for any other purpose desired by the XSL code.

In this case, the XSL code does use this number to identify the set of records that are generated for a single message, but with a twist.

The goal is for an iterating SubPlan within the Data Flow to iterate once for each set of records produced by a single message, using the Iterate each time a key field changes value option for the SubPlan. The XSL passes the sequence number in the Data Flow records as a field named "seq_no", and this is the key field used for SubPlan iteration.

The SubPlan does not actually iterate until the key value changes. If a gap in time exists between messages, completion of processing for the prior message is delayed until the next message is received since changes are not committed until the SubPlan iterates through using the "Group Transaction" feature. To solve this problem, the XSL always generates an extra "dummy" record, or tag, at the end of the set of records for each message. Since the sequence numbers are incremented by one, the XSL can know what the next sequence number will be, unlike with an MSMQ message ID. It can use the sequence number that the next message will use. The Data Flow contains logic to recognize and ignore the dummy records, which are denoted by an "ord_id" attribute value of "dummy".

36.3.2.5 XSLT Code

[Example 36-1](#) shows the XSLT code contained within the Enterprise Message Source.

Example 36-1 Enterprise Message Source XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="SequenceNumber"/>
  <xsl:output method="xml" omit-xml-declaration="yes"/>
  <xsl:template match="/">
    <PhoenixDebtOrder>
      <xsl:choose>
        <xsl:when test="PhoenixDebtOrder">
          <xsl:apply-templates select="PhoenixDebtOrder"/>
        </xsl:when>
        <xsl:otherwise>
          <!-- There is no root PhoenixDebtOrder tag -->
          <!-- Generate a tag with no attribute values -->
          <Flat>
            <xsl:attribute name="seq_no">
              <xsl:value-of select="$SequenceNumber"/></xsl:attribute>
          </Flat>
        </xsl:otherwise>
      </xsl:choose>
      <!-- Dummy row, always generated at the end, with the sequence number
           that the next set of rows will have. Used to cause the SubPlan
           to iterate immediately after the receipt of all rows for this order-->
      <Flat ord_id="Dummy">
        <xsl:attribute name="seq_no">
          <xsl:value-of select="$SequenceNumber + 1"/></xsl:attribute>
      </Flat>
    </PhoenixDebtOrder>
  </xsl:template>
```

```

<xsl:template match="PhoenixDebtOrder">
  <xsl:choose>
    <xsl:when test="PhoenixDebtOrderProduct">
      <xsl:apply-templates select="PhoenixDebtOrderProduct"/>
    </xsl:when>
    <xsl:otherwise>
      <!-- There are no products -->
      <!-- Generate a tag with only the attributes from the root
            PhoenixDebtOrder tag -->
      <Flat>
        <xsl:for-each select="@*">
          <xsl:copy-of select="."/>
        </xsl:for-each>
        <xsl:attribute name="seq_no">
          <xsl:value-of select="$SequenceNumber"/></xsl:attribute>
        </Flat>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
<xsl:template match="PhoenixDebtOrderProduct">
  <xsl:choose>
    <xsl:when test="PhoenixDebtOrderIOI">
      <xsl:apply-templates select="PhoenixDebtOrderIOI"/>
    </xsl:when>
    <xsl:otherwise>
      <!-- This product has no IOIs -->
      <!-- Generate a tag with only the attributes from the root
            PhoenixDebtOrder tag and the parent PhoenixDebtOrderProductTag -->
      <Flat>
        <xsl:for-each select="../@*">
          <xsl:copy-of select="."/>
        </xsl:for-each>
        <xsl:for-each select="@*">
          <xsl:copy-of select="."/>
        </xsl:for-each>
        <xsl:attribute name="seq_no">
          <xsl:value-of select="$SequenceNumber"/></xsl:attribute>
        </Flat>
        <!-- Add a linebreak to the output XML, just for appearance -->
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
<xsl:template match="PhoenixDebtOrderIOI">
  <Flat>
    <!-- Root PhoenixDebtOrder attributes -->
    <xsl:for-each select="../../@*">
      <xsl:copy-of select="."/>
    </xsl:for-each>
    <!-- PhoenixDebtOrderProduct attributes -->
    <xsl:for-each select="../@*">
      <xsl:copy-of select="."/>
    </xsl:for-each>
    <!-- PhoenixDebtOrderIOI attributes -->
    <xsl:for-each select="@*">
      <xsl:copy-of select="."/>
    </xsl:for-each>
    <xsl:attribute name="seq_no">
      <xsl:value-of select="$SequenceNumber"/></xsl:attribute>
    </Flat>
  </xsl:template>

```

```
<!-- Add a linebreak to the output XML, just for appearance -->
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

36.4 Editing Enterprise Message Sources

To edit an enterprise message source:

1. Select **Enterprise Message Sources** from the Architect function list.
2. Click the name of the enterprise message source.
The message source properties display.
3. Click **Edit**.
4. Make the changes and click **Save**.

36.5 Copying Enterprise Message Sources

To copy an enterprise message source:

1. Select **Enterprise Message Sources** from the Architect function list.
2. Click the name of the enterprise message source to copy.
The message source properties display.
3. Click **Copy**.
4. Type a new name for the copy of the enterprise message source and click **Copy**.
The new message source is created and added to the list.

36.6 Deleting Enterprise Message Sources

To delete an enterprise message source:

1. Select **Enterprise Message Sources** from the Architect function list.
2. Click the name of the enterprise message source to delete.
The message source properties display.
3. Click **Delete**.
4. Click **OK** to confirm that you want to delete the message source.
The message source is deleted.

Using ICommand

This chapter provides usage and reference material for the ICommand command-line utility and web service. It contains the following topics:

- [Section 37.1, "Introducing ICommand"](#)
- [Section 37.2, "Executing ICommand"](#)
- [Section 37.3, "General Command and Option Syntax"](#)
- [Section 37.4, "Object Name Syntax"](#)
- [Section 37.5, "Command-line-only Parameters"](#)
- [Section 37.7, "Summary of Individual Commands"](#)
- [Section 37.8, "Detailed Command Descriptions"](#)
- [Section 37.9, "Format of Command File"](#)
- [Section 37.10, "Format of Log File"](#)
- [Section 37.11, "Sample Export File"](#)
- [Section 37.12, "Regular Expressions"](#)
- [Section 37.13, "Using ICommand Web Service"](#)

37.1 Introducing 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.

37.2 Executing ICommand

Before attempting to execute ICommand, the `JAVA_HOME` environment variable must be set to point to the root directory of the supported version of JDK (JDK 1.5 or later).

ICommand can be executed using the `SOA_ORACLE_HOME\bin\icommand.bat` file on the Microsoft Windows platform and `SOA_ORACLE_HOME\bin\icommand` shell script on Unix platforms.

37.3 General Command and Option Syntax

The basic structure of the ICommand command line entry is as follows:

```
icommand -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 must contain spaces or other special characters, it may be enclosed in double quotes. For example

```
icommand -cmd export -name "/Samples/Call center" -type dataobject -file  
C:\Callcenter.xml
```

It is required to use quotes around report names and file names that contain spaces and other special characters.

For some parameters, the *value* may be omitted.

37.3.1 Specifying 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.

37.4 Object Name Syntax

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:

```
/My Folder/My Subfolder/My Data Object
```

If the Data Object is at the root, the leading slash (/) is optional. The following two examples are equivalent:

```
/My Data Object  
My Data Object
```

Reports

The full path name must be specified as in the following examples.

For shared reports:

```
"/public/Report/Subfolder1/My Report"
```

For private reports:

```
"/private:user_name/Report/Subfolder1/My Report"
```

For private reports the `/private:user_name/` prefix may be omitted if the user running ICommand is the user that owns the report.

The path information without the `public` or `private` prefix is saved in the export file.

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.

37.5 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 filename
```

Optional parameter that specifies the name of the file that contains commands to be processed. Since 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 in the event of 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 will be 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 machine on which the Active Data Cache server is running).

If this parameter is omitted, `main` is used, which means the server information will be obtained from the `ADCServerName` key in the `ICommand.exe.config` file.

If the reserved value `ADCInProcServer` is used, then `ICommand` will directly access the Active Data Cache database (which must be local on the same machine on which `ICommand` is running) rather than contacting the Active Data Cache server. This option should be used **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 filename`

Optional parameter that specifies the name of the file to which results and errors are logged. If the file does not exist, it will be created. If the file does exist, any contents will be overwritten. Since 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 will be output to the console.

- 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 will be 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 will be no top level root tag in the XML produced by successive appends (`ICommand` will append the same tag each time it is run). It is left up to the user to handle this.

- Username

`-username user_name`

- Password

`-password password`

37.6 Running ICommand Remotely

You can run `ICommand` from a remote machine (where Oracle BAM is installed) and execute the commands on a server located remotely. To do this, we need to add the

properties `ServerName` and `ServerPort` in `SOA_ORACLE_HOME\config\BAMICCommandConfig.xml`, as shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns0:BAMICCommand xmlns:ns0="http://xmlns.oracle.com/bam/configuration/server/icommand"
  SensorFactory="oracle.bam.common.statistics.noop.SensorFactoryImpl"
  GenericSatelliteChannelName="invm:topic/oracle.bam.messaging.systemobjectnotification"
  ServerName="host_name"
  ServerPort = "9085"
</ns0:BAMICCommand>
```

The Oracle BAM version installed on the remote machine should be same as the Oracle BAM server version (that is, both servers should be from the same label).

37.7 Summary of Individual Commands

[Table 37-1](#) summarizes the commands.

Table 37-1 ICommand Command Summary

Command	Parameters
export	-file <i>filename</i> [-name <i>itemname</i>] [-type [dataobject folder report rule securityfilters user distributionlist ems eds all]] [-match <i>pattern</i>] [-regex <i>regularexpression</i>] [-all [0 1]] [-systemobjects [0 1]] [-dependencies [0 1]] [-layout [0 1]] [-contents [0 1]] [-permissions [0 1]] [-privileges [0 1]] [-members [0 1]] [-owner [0 1]] [-header [0 1]] [-footer [0 1]] [-append [0 1]] [-preview [0 1]]

For more information about `export` see [Section 37.8.1, "Export."](#)

Table 37–1 (Cont.) ICommand Command Summary

Command	Parameters
import	-file <i>filename</i> -continueonerror [-delay <i>milliseconds</i>] [-updatelayout] [-mode [preserveid update overwrite append rename error]] [-preserveowner] [-setcol <i>col_name</i> /[null now value: <i>override_value</i>]] For more information about import see "Import" on page 37-10.
delete	[-name <i>itemname</i>] [-type [dataobject folder report rule securityfilters user distributionlist ems eds all]] [-match <i>pattern</i>] [-regex <i>regularexpression</i>] [-all [0 1]] [-systemobjects [0 1]] For more information about delete see Section 37.8.3, "Delete."
rename	-name <i>itemname</i> -newname <i>newitemname</i> [-type [dataobject folder report rule user distributionlist ems eds all]] For more information about rename see Section 37.8.4, "Rename."
clear	-name <i>itemname</i> [-type [dataobject folder distributionlist]] For more information about clear see Section 37.8.5, "Clear."

37.8 Detailed Command Descriptions

This section details each of the ICommand commands, their parameters, and gives examples.

37.8.1 Export

Exports information about one or more objects in the Active Data Cache to an XML file. See [Section 37.11, "Sample Export File"](#) for an example of an exported Data Object.

Table 37–2 Export Command Parameters

Parameter	Description
-file <i>filename</i>	The name of the file to export to. Required. If the file does not exist, it will be created. If the file does exist, any contents will be overwritten, unless the <code>append</code> parameter is used. Since the file will contain XML, it would usually have an XML extension.
-name <i>itemname</i>	The name of the item to be exported.

Table 37-2 (Cont.) Export Command Parameters

Parameter	Description
<code>-type <i>itemtype</i></code>	<p>The type of the item to be exported. Must be one of the following:</p> <ul style="list-style-type: none"> ▪ <code>dataobject</code> ▪ <code>folder</code> ▪ <code>report</code> ▪ <code>rule</code> ▪ <code>securityfilters</code> (For the specified Data Objects) ▪ <code>user</code> ▪ <code>distributionlist</code> ▪ <code>ems</code> (Enterprise Message Source) ▪ <code>eds</code> (External Data Source) ▪ <code>all</code> <p><code>dataobject</code> is assumed if this parameter is omitted.</p>
<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 will be exported.
<code>-regex <i>regexexpr</i></code>	A regular expression pattern matching string. The items whose names match the pattern will be exported. See Section 37.12, "Regular Expressions" for more information.
<code>-all [0 1]</code>	<p>Controls whether all items of the specified type will be 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>
<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>-dependencies [0 1]</code>	<p>Applies only to Data Objects. Controls whether other Data Objects that the exported Data Objects depend on in the lookup columns will also be 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 will also be exported.</p>
<code>-layout [0 1]</code>	<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>
<code>-contents [0 1]</code>	<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>

Table 37-2 (Cont.) Export Command Parameters

Parameter	Description
-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. nonzero 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>
-privileges [0 1]	<p>Applies only to Roles. Controls whether the privilege settings in the Roles being exported are exported.</p> <p>A nonzero value means export the privilege settings. Zero (0) means do not export the privilege settings. nonzero is assumed if this parameter is omitted, or if the value is omitted.</p>
-members [0 1]	<p>Applies only to Roles. Controls whether the list of users in the Roles being exported are exported.</p> <p>A nonzero value means export the list of users. Zero (0) means do not export the list of users. nonzero is assumed if this parameter is omitted, or if the value is omitted.</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>
-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>
-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>
-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. nonzero is assumed if this parameter is omitted, or if the value is omitted.</p>

Table 37-2 (Cont.) Export Command Parameters

Parameter	Description
<code>-preview [0 1]</code>	<p>In preview mode, ICommand 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, ICommand export continues processing even after some errors that would cause non-preview mode to stop the export.</p> <p>A nonzero value means preview mode. nonzero is assumed if the value is omitted. Zero (0) is assumed if the parameter is omitted.</p>

Example 37-1 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 37-2 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 37-3 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 37-4 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 37-5 Exporting a Folder from Data Objects

```
icommand -cmd export -name "/public/DataObject/Test Sub folder" -type folder -file
C:\foldertest1.xml
```

Example 37-6 Exporting a Private Report

As in [Example 37-3](#), 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 37-7 Exporting a Shared Report

```
icommand -cmd export -name "/public/Report/SharedReport" -type report -file  
C:\SharedReport.xml
```

Example 37-8 Exporting All of the Reports in the System

```
icommand -cmd export -type report -all -file C:\temp\TestAll.xml
```

Example 37-9 Exporting an Alert Rule

```
icommand -cmd export -name Alert1 -type rule -file C:\Alert1.xml
```

Example 37-10 Exporting a Security Filter

```
icommand -cmd export -type securityfilters -name "TestDO" -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 37-11 Exporting a User

```
icommand -cmd export -name bamadmin -type user -file C:\TestUser.xml
```

Example 37-12 Exporting a Distribution List

```
icommand -cmd export -name MyDistList -type distributionlist -file  
C:\MyDistList.xml
```

Example 37-13 Exporting an Enterprise Message Source

```
icommand -cmd export -type ems -name TestEMS -file C:\TestEMS.xml
```

Example 37-14 Exporting an External Data Source

```
icommand -cmd export -type eds -name TestEDS -file C:\TestEDS.xml
```

Example 37-15 Exporting All Oracle BAM System Objects

```
icommand -cmd export -type all -file C:\temp\TestAll.xml
```

37.8.2 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 will be created if possible. For Data Objects, the input file must contain layout information in order to create the Data Object, and if the file contains no content information, then an empty Data Object will be 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 37-3 Import Command Parameters

Parameter	Description
<code>-file filename</code>	The name of the file to import from. Required. This would usually be a file that was created through the export command.
<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.
<code>-delay millisec</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 will be no delay.
<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. nonzero is assumed if this parameter is omitted, or if the value is omitted.
<code>-updatelayout</code>	Applies only to Data Objects. Controls whether, if the Data Object being imported already 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.

Table 37-3 (Cont.) Import Command Parameters

Parameter	Description
<code>-mode mode</code>	<p>The following mode values are valid for Folders, Reports, Users, EMS, and EDS objects:</p> <ul style="list-style-type: none"> ■ <code>overwrite</code> If the item already exists, replace it with the imported item. ■ <code>rename</code> If the item already exists, change the name of the imported item. The new name is computed automatically and reported in a message. ■ <code>error</code> If the item already exists, terminate 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 already exists, replace it with the imported item. ■ <code>rename</code> If the item already exists, change the name of the imported item. The new name is computed automatically and reported in a message. ■ <code>append</code> If the item already exists, append the users in the imported list to the already existing list. ■ <code>error</code> If the item already exists, terminate the import with an error. <p>Only the following values are valid for Data Objects:</p> <ul style="list-style-type: none"> ■ <code>preserveid</code> If the imported Data Object does not already exist and must be created, ICommand will attempt to assign the Data Object the same internal ID that the exported Data Object had. If it is unable to, the import will be terminated with an error. This option is important because some other items, such as Reports, point to the Data Objects they use by ID, not by name. ■ <code>update</code> Normally, 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 already contains at least one Security Filter, the import will be aborted with an error.</p> <p>This parameter is not supported for Rules.</p>

Table 37–3 (Cont.) Import Command Parameters

Parameter	Description
<code>-setcol</code>	<p>Allows override of column values from the command line during import, including setting to current date/time.</p> <pre>-setcol <i>column_name</i>/NULL -setcol <i>column_name</i>/NOW -setcol <i>column_name</i>/VALUE:<i>override-value</i></pre> <p><i>column_name</i> is the name of one of the columns 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, as long as 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 will be 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 at the time that 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:<i>override-value</i> 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>In order to allow more than one column to be overridden, any number of <code>setcol</code> parameters may be present. However, since duplicate parameters are not permitted, ICommand will recognize 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>Example command line:</p> <pre>icommand -cmd import -file myfile.xml -setcol1 Field1/null -setcol2 Field3/now -setcol3 "Customer Name/value:John Q. Public"</pre>

Example 37–16 Importing a Data Object With Delay

```
icommand -cmd import -file C:\TestDO.xml -delay 1000 -continueonerror 1
```

37.8.3 Delete

Deletes an item from the Active Data Cache.

Table 37–4 Delete Command Parameters

Parameter	Description
<code>-name <i>itemname</i></code>	The name of the item to be deleted.

Table 37–4 (Cont.) Delete Command Parameters

Parameter	Description
<code>-type <i>itemtype</i></code>	The type of the item to be deleted. Must be one of the following: <ul style="list-style-type: none"> ▪ <code>dataobject</code> ▪ <code>folder</code> ▪ <code>report</code> ▪ <code>rule</code> ▪ <code>securityfilters</code> (For the specified Data Objects) ▪ <code>user</code> ▪ <code>distributionlist</code> ▪ <code>ems</code> (Enterprise Message Source) ▪ <code>eds</code> (External Data Source) ▪ <code>all</code> <code>dataobject</code> 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 will be deleted.
<code>-regex <i>regularexpr</i></code>	A regular expression pattern matching string. The items whose names match the pattern will be deleted. See Section 37.12, "Regular Expressions" for more information.
<code>-all [0 1]</code>	Controls whether all items of the specified type will be deleted. 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>-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.

Example 37–17 Deleting a Data Object

```
icommand -cmd delete -type dataobject -name TestDO
```

37.8.4 Rename

Renames an item in the Active Data Cache.

Table 37–5 Rename Command Parameters

Parameter	Description
<code>-name <i>itemname</i></code>	The name of the item to be renamed. Required.
<code>-newname <i>newitemname</i></code>	The new name for the item. Required. For Data Objects, Reports and Folders, only the new base name should be given, with no path (for example MyReport).

Table 37–5 (Cont.) Rename Command Parameters

Parameter	Description
<code>-type <i>itemtype</i></code>	The type of object to be renamed. Must be one of the following: <ul style="list-style-type: none"> ▪ <code>dataobject</code> ▪ <code>folder</code> ▪ <code>report</code> ▪ <code>rule</code> ▪ <code>user</code> ▪ <code>distributionlist</code> ▪ <code>ems</code> (Enterprise Message Source) ▪ <code>eds</code> (External Data Source) <code>dataobject</code> is assumed if this parameter is omitted.

Example 37–18 Renaming a Distribution List

```
icommand -cmd rename -type distributionlist -name TestList -newname MyDistList
```

37.8.5 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) are removed from the distribution list.

Table 37–6 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. Must be one of the following: <ul style="list-style-type: none"> ▪ <code>dataobject</code> ▪ <code>folder</code> ▪ <code>distributionlist</code> <code>dataobject</code> is assumed if this parameter is omitted.

Example 37–19 Clearing a Data Object

```
icommand -cmd clear -name "/Samples/Call Center" -type dataobject
```

37.9 Format of Command File

This section contains the following topics:

- [Section 37.9.1, "Inline Content"](#)
- [Section 37.9.2, "Command IDs"](#)
- [Section 37.9.3, "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>
```

37.9.1 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" encoding="utf-8"?>
<OracleBAMCommands>
<Import inline="1">
<Export Version="504.0" Build="3.0.3697.0">
  <DataObject Version="13" Name="Employees" ID="_Employees" Path="/Samples"
  External="0">
    <Layout>
      <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
      Nullable="1" />
      <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
      Nullable="1"/>
      <Column Name="Sales Number" ID="_Sales_Number" Type="integer" Nullable="1"
      />
      <Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0" />
    <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="2004-09-14T14:07:41.5600000-07:00" />
    </Row>
  </Contents>
</DataObject>
</Export>
</Import>
</OracleBAMCommands>
```

37.9.2 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 will be included in the Result or Error elements of any output related to that command.

Sample Input:

```
<OracleBAMCommands>
  <Delete id="1" type="role" name="Report Creator" />
  <Delete id="2" type="user" name="joeschmoe" />
</OracleBAMCommands>
```

Sample Output:

```
<ICommandLog Login="MSOLNIT-PC\ASPNET">
  <Results Command="Delete" ID="1">Role "Report Creator" deleted.</Results>
  <Error Command="Delete" ID="2">
    <![CDATA[Error while processing command "Delete".
[ErrorSource="ICommandEngine", ErrorID="ICommandEngine.Error"] There is no User
named "joeschmoe". [ErrorSource="ICommandEngine",
ErrorID="ICommandEngine.UserExist"]]]>
  </Error>
</ICommandLog>
```

37.9.3 Continue On Error

Ordinarily, ICommand will execute 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 will be executed. This behavior can be changed by using the `continueonerror` attribute at either a global level or for each command.

[Example 37–20](#) shows how to use the `continueonerror` attribute so that all commands will be executed regardless of if any failures occur

Example 37–20 Enabling Global Continue-On-Error

```
<OracleBAMCommands continueonerror="1">
  <Delete id="1" type="role" name="Report Creator" />
  <Delete id="2" type="user" name="joeschmoe" />
</OracleBAMCommands>
```

In [Example 37–21](#), `continueonerror` only applies to the command that deletes user `joeschmoe`. If this command fails, then ICommand will output the error and continue. But if any other command fails, ICommand will immediately stop.

Example 37–21 Enabling Command Continue-On-Error

```
<OracleBAMCommands>
  <Delete id="1" type="role" name="Report Creator" />
  <Delete id="2" type="user" name="joeschmoe" continueonerror="1" />
  <Delete id="3" type="user" name="user2" />
  <Delete id="4" type="user" name="user3" />
</OracleBAMCommands>
```

37.10 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.

Sample log file (output from the preceding sample command file):

```

<?xml version="1.0" encoding="utf-8"?>
<ICommandLog Login="MYDOMAIN\myaccount">
  <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.EMSExist"]]]></Error>
</ICommandLog>

```

37.11 Sample Export File

The following example shows a sample file resulting from exporting a Data Object.

```

<?xml version="1.0" encoding="utf-8"?>
<OracleBAMExport Version="504.0" Build="3.0.3697.0">
  <DataObject Version="13" Name="Employees" ID="_Employees" Path="/Samples"
  External="0">
    <Layout>
      <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
      Nullable="1" />
      <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
      Nullable="1" />
      <Column Name="Sales Number" ID="_Sales_Number" Type="integer"
      Nullable="1" />
      <Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0" />
    </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="2004-09-14T14:07:41.5600000-07:00" />
      </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="2004-09-14T14:07:41.5600000-07:00" />
      </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="2004-09-14T14:07:41.5600000-07:00" />
      </Row>
    </Contents>
  </DataObject>
</OracleBAMExport>

```

37.12 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.

The following table contains the complete list of metacharacters and their behavior in the context of regular expressions:

Table 37–7 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 "fooooo". <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 "fooooo". <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> .

Table 37–7 (Cont.) Metacharacters for Regular Expressions

Character	Description
<code>(pattern)</code>	A subexpression that matches <i>pattern</i> and captures the match. The captured match can be retrieved from the resulting Matches collection using the \$0 . . . \$9 properties. To match parentheses characters (), use '\(' or '\)'. <i>(pattern)</i>
<code>(?:pattern)</code>	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, 'industr(?:y ies)' is a more economical expression than 'industry industries'. <i>(?:pattern)</i>
<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. <i>(?=pattern)</i>
<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. <i>(?!pattern)</i>
<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 one 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.

Table 37-7 (Cont.) Metacharacters for Regular Expressions

Character	Description
<code>\n</code>	Matches a newline character. Equivalent to <code>\x0a</code> and <code>\cJ</code> .
<code>\r</code>	Matches a carriage return character. Equivalent to <code>\x0d</code> and <code>\cM</code> .
<code>\s</code>	Matches any white space character including space, tab, form-feed, and so on. Equivalent to <code>[\f\n\r\t\v]</code> .
<code>\S</code>	Matches any non-white space character. Equivalent to <code>[^\f\n\r\t\v]</code> .
<code>\t</code>	Matches a tab character. Equivalent to <code>\x09</code> and <code>\cI</code> .
<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' & "1"</code> . 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 (©).

37.13 Using ICommand Web Service

ICommand is available as a web service for application developers who want to interact with ICommand features over HTTP.

This section contains the following topics:

- [Section 37.13.1, "Differences between the ICommand Web Service and the ICommand Command-Line Utility"](#)
- [Section 37.13.2, "Using the ICommand Web Service"](#)
- [Section 37.13.3, "Security Issues"](#)

37.13.1 Differences between the ICommand Web Service and the ICommand Command-Line Utility

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 will receive back the `export` content inline.

Commands other than `import` and `export` generally work the same as with the command-line utility.

37.13.2 Using the ICommand Web Service

The ICommand web service is available on the machine where Report Server has been installed. It is available at the following URL

```
http://host_name:port_number/OracleBAM/WebServices/ICommand
```

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 37.9, "Format of Command File."](#) The return value is a string containing the results of executing each command, in the log syntax described in [Section 37.10, "Format of Log File."](#)

Example 37–22 Exporting a Data Object (Input)

```
<OracleBAMCommands>
  <Export name='/Samples/Film Sales' inline='1' />
</ OracleBAMCommands>
```

Example 37–23 Exporting a Data Object (Output)

```
<ICommandLog Login="MSOLNIT-PC\ASPNET">
  <OracleBAMExport Version="1003.0" Build="3.5.5603.0">
    <DataObject Version="14" Name="Film Sales" ID="_Film_Sales"
      Path="/Samples" External="0">
      <Layout>
        <Column Name="Region" ID="_Region" Type="string" MaxSize="100"
          Nullable="1" Public="1" />
        <Column Name="State" ID="_State" Type="string" MaxSize="100" Nullable="1"
          Public="1" />
        <Column Name="Category" ID="_Category" Type="string" MaxSize="100"
          Nullable="1" Public="1" />
        <Column Name="Brand" ID="_Brand" Type="string" MaxSize="100" Nullable="1"
          Public="1" />
        <Column Name="Description" ID="_Description" Type="string" MaxSize="100"
          Nullable="1" Public="1" />
        <Column Name="Sales" ID="_Sales" Type="integer" Nullable="1" Public="1" />
      <Indexes />
    </Layout>
  </Contents>
```

```

<Row ID="1">
  <Column ID="_Region" Value="Western Region" />
  <Column ID="_State" Value="Arizona" />
  <Column ID="_Category" Value="Film" />
  <Column ID="_Brand" Value="Kodak" />
  <Column ID="_Description" Value="35mm 200" />
  <Column ID="_Sales" Value="2000" />
</Row>
<Row ID="2">
  <Column ID="_Region" Value="Western Region" />
  <Column ID="_State" Value="Arizona" />
  <Column ID="_Category" Value="Film" />
  <Column ID="_Brand" Value="Kodak" />
  <Column ID="_Description" Value="35mm 400" />
  <Column ID="_Sales" Value="2100" />
</Row>
</Contents>
</DataObject>
</OracleBAMExport>
<Results Command="Export">Exporting Data Object "/Samples/Film
Sales"...</Results>
<Results Command="Export">Data Object "/Samples/Film Sales"
exported successfully (29 rows).</Results>
<Results Command="Export">1 items exported successfully.</Results>
</ICommandLog>

```

37.13.3 Security Issues

The following are security issues with using the ICommand web service.

37.13.3.1 IIS Security (HTTP 401 error)

The Web server might not be configured to allow anonymous access to the Web service. In this case, you must include credentials in the Web service call. The method for doing this varies depending on your host environment (.NET, Java, and so on).

37.13.3.2 Active Data Cache Security

If the Web server does allow anonymous access, then it will typically run using a low-privilege account (such as the local ASPNET account). This account may not have access to all Oracle Business Activity Monitoring features, such as creating users or manipulating data objects.

Oracle User Messaging Service

This part describes how to use the Oracle User Messaging Service.

This part contains the following chapters:

- [Chapter 38, "Configuring User Messaging Service,"](#)
- [Chapter 39, "Parlay X Web Services Multimedia Messaging API"](#)
- [Chapter 40, "User Messaging Preferences"](#)
- [Chapter 41, "Send Message to User Specified Channel Sample Application"](#)
- [Chapter 42, "Oracle User Messaging Service Expense Report Use Case"](#)

Configuring User Messaging Service

This chapter describes how to configure Oracle User Messaging Service (UMS) to support SMS, Voice, and IM channels. This chapter includes the following sections:

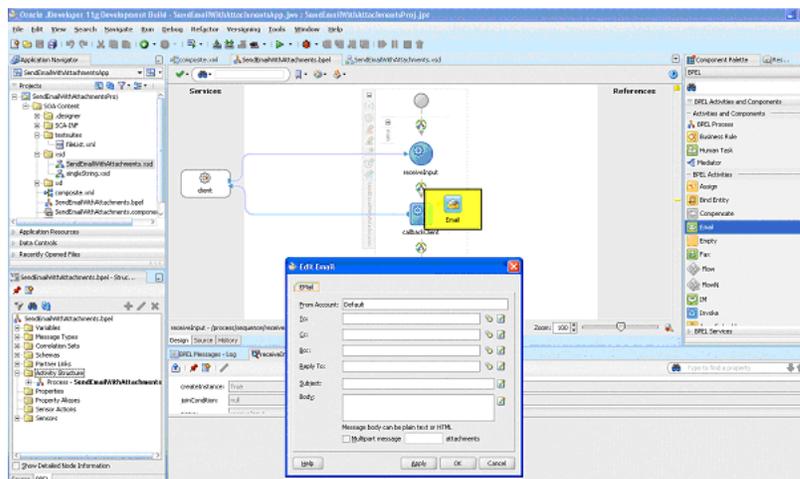
- Section 38.1, "Overview of Oracle User Messaging Service Configuration"
- Section 38.2, "Configuring User Messaging Service Drivers"
- Section 38.3, "Deploying Drivers"
- Section 38.4, "Configuring Messaging Preferences"

38.1 Overview of Oracle User Messaging Service Configuration

Oracle User Messaging Service enables users to receive notifications sent from SOA applications that are developed and deployed to the Oracle 11g application server using Oracle JDeveloper.

At the application level, the BPEL process includes a notification activity for a specific delivery channel, such as SMS or E-Mail. For example, when you build a SOA application that sends e-mail notification, you drag and drop an *Email Activity* component from the JDeveloper *Component Palette* to the appropriate location within the BPEL workflow (Figure 38–1). BPEL connects with Oracle User Messaging Service to send notifications.

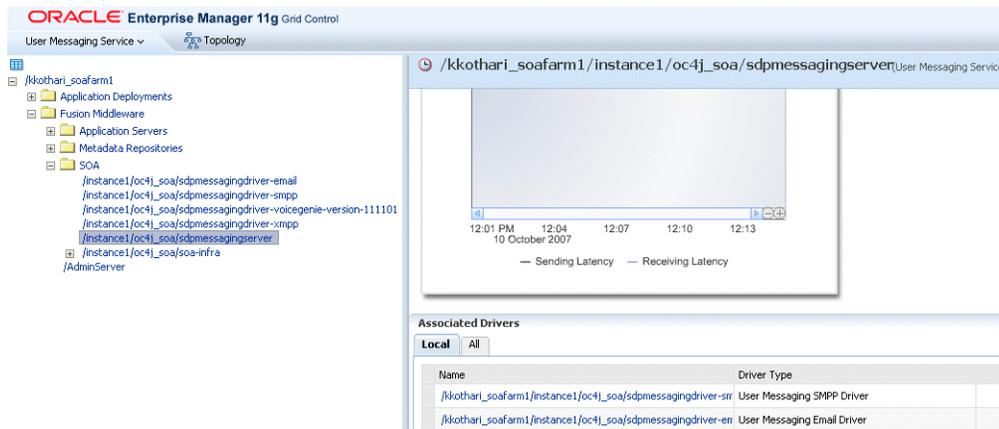
Figure 38–1 Setting the Notification Activity in the BPEL Workflow



To enable the workflow participants to receive and forward notifications, use Oracle 11g Enterprise Manager to set the Oracle User Messaging Service environment by

configuring the appropriate driver instances that reside on the same application server on which you deploy the workflow application (Figure 38–2). Oracle User Messaging Service ships with drivers that support messaging through the E-Mail, IM, SMS and voice channels. For more information, see Section 38.2, "Configuring User Messaging Service Drivers".

Figure 38–2 Oracle Enterprise Manager 11g Grid Control



For the workflow participants to actually receive the notifications, they must register the devices that they use to access messages through User Messaging Preferences (Figure 38–3). For more information, see Section 38.4, "Configuring Messaging Preferences".

Figure 38–3 User Messaging Preferences



38.2 Configuring User Messaging Service Drivers

Oracle User Messaging ships with the following drivers.

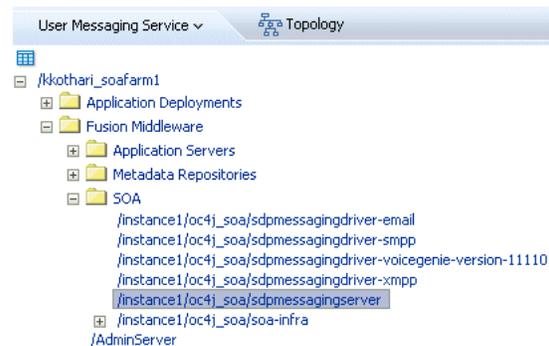
- E-Mail Driver (See Section 38.2.2)
- SMPP Driver (See Section 38.2.3)
- XMPP Driver (See Section 38.2.4)
- VoiceGenie Driver (See Section 38.2.5)
- Proxy Driver (See Section 38.2.6)

Note: Oracle User Messaging Service ships with only the E-Mail driver deployed. For information on deploying the other drivers, see [Section 38.3, "Deploying Drivers"](#).

To configure a driver:

1. Log into the Enterprise Manager Fusion Middleware Control console as an administrator.
2. Expand the *Fusion Middleware* folder and then the *SOA* folder ([Figure 38-4](#)).

Figure 38-4 Expanding the SOA Folder



3. Navigate to the User Messaging Service *Home* page (*/instance/oc4j_soa/sdpmessagingserver* in [Figure 38-4](#)).
4. If needed, expand the *Associated Drivers* section ([Example 38-5](#)) of the User Messaging Service *Home* page.

Figure 38-5 Drivers Associated with the SOA Instance

Associated Drivers			
Local All			
Name	Driver Type	Status	Configure Driver
/kkothari_soafarm1/instance1/oc4j_soa/sdpmessagingdriver-smpp	User Messaging SMPP Driver		
/kkothari_soafarm1/instance1/oc4j_soa/sdpmessagingdriver-email	User Messaging Email Driver		
/kkothari_soafarm1/instance1/oc4j_soa/sdpmessagingdriver-voicegenie-version-	User Messaging VoiceGenie Driver		
/kkothari_soafarm1/instance1/oc4j_soa/sdpmessagingdriver-xmpp	User Messaging XMPP Driver		

5. Select the *Local* tab to access the driver deployed to the SOA instance. The *ALL* tab lists drivers that are deployed to all of the instances.
6. Select a driver (such as the E-Mail driver, illustrated in [Figure 38-5](#)) and then click the adjacent **Edit** icon (illustrated in [Example 38-6](#) as a pencil) in the *Configure Driver* column.

Figure 38-6 The Edit Icon



The configuration page displays ([Figure 38-7](#)).

Figure 38–7 The Basic Configuration Page for a Selected Driver

Revert Apply

Basic Configuration

Common Configuration

Supported Delivery Types: EMAIL * Capability: BOTH Cost: [dropdown] Speed: [dropdown] Sender Addresses: [text] Default Sender Address: [text]	Supported Protocols: [text] Supported Carriers: [text] Supported Content Types: text/plain, text/html, multipart/mixed, multipart/alternative, multipart/related Supported Status Types: DELIVERY_TO_GATEWAY_SUCCESS, DELIVERY_TO_GATEWAY_FAILURE, USER_REPLY_ACKNOWLEDGEMENT_SUCCESS, USER_REPLY_ACKNOWLEDGEMENT_FAILURE Sending Queues Info: OraSDPM/QueueConnectionFactory:OraSDPM/Queues/OraSDPMDriverDefSndQ1
--	--

Driver-Specific Configuration

Name	Description	Mandatory	Value
MailAccessProtocol	E-mail receiving protocol. The possible values are IMAP and POP3. Required only if e-mail receiving is supported on the driver instance	[checkbox]	IMAP
RetryLimit	This value specifies the number of times to retry connecting to the incoming mail server, if the connection is lost due to some reason. The default value is -1 which means no limit to the number of tries.	[checkbox]	-1

7. If needed, expand the *Driver-Specific Configuration* section and configure the driver parameters. For more information, see [Section 38.2.1, "Configuring Driver Properties"](#).

38.2.1 Configuring Driver Properties

Oracle User Messaging Service drivers share common properties (listed in [Table 38–1](#)) that are used by the Messaging Engine when routing outbound messages. Typically, administrators set such Quality of Service (QoS) properties as driver cost (*Cost*) and driver speed (*Speed*), supported carriers (*SupportedCarriers*), and supported protocols (*SupportedProtocols*). Driver developers configure properties that typically do not require modification by the administrator, such as supported delivery types (*SupportedDeliveryTypes*), and supported content types (*SupportedContentTypes*).

Note: Properties such as *SendingQueuesInfo* are for advanced use and only require modification for advanced deployment topologies.
[[What are others??]]

Table 38–1 Common Driver Properties

Name	Description	Mandatory Property?	Default Value(s)
Capability	Sets the driver's capability to send or receive messages. The values are <i>SEND</i> , <i>RECEIVE</i> , and <i>BOTH</i> .	Yes	The default value is <i>BOTH</i> .
Cost	The cost level of the the driver (from 0 - 10). 0 is least expensive; 10 is most expensive. If the value is not in this range, cost is considered to be 0.	No	N/A
DefaultSenderAddress	The default address of the sender. The driver uses these addresses when sending a message that has no sender address specified, or when the specified sender address is not in the sender addresses list and the driver does not support using the application-provided sender address.	No	N/A
InstanceName	The name of the driver instance.	Yes	The instance name must be unique.
SenderAddresses	The list of sender addresses that the driver supports. If provided by the driver, the Messaging Engine can use this to route a sending message to the driver by matching against the sender address of the message.	No	N/A
SendingQueuesInfo	The information for the Driver Sending Queue.	Yes	The default value is: <i>OracleNS/QueueConnectionFactory:OracleNS/Queus/OracleSDPMDriverDefSndQ1 [!?!]</i>
Speed	The speed level of the driver (from 0-10).	No	N/A
SupportedCarriers	A comma-separated list of supported carriers.	No	N/A
SupportedContentTypes	The content type supported by the driver.	Yes	Depending on the driver type, the default content types include <i>text/plain</i> , <i>text/html</i> , <i>multipart/mixed</i> , <i>multipart/alternative</i> , <i>multipart/related</i> .

Table 38–1 (Cont.) Common Driver Properties

Name	Description	Mandatory Property?	Default Value(s)
SupportedDeliveryTypes	The delivery types supported by the driver.	Yes	Enter one, or a combination of, the following values: <ul style="list-style-type: none"> ■ SMS ■ IM ■ Voice ■ E-Mail
SupportedProtocols	A comma-separated list of supported protocols. Entering an asterisk (*) for any protocol.	No	N/A
SupportedStatusTypes	The status types supported by the driver.	No	Default values include: <ul style="list-style-type: none"> ■ DELIVERY_TO_GATEWEAY_SUCCESS ■ DELIVERY_TO_GATEWAY_FAILURE ■ USER_REPLY_ACKNOWLEDGEMENT_SUCCESS ■ USER_REPLY_ACKNOWLEDGEMENT_FAILURE
SupportsCancel	Supports a Cancel operation on a message.	No	The default value is <i>false</i> .
SupportsReplace	Supports a Replace operation on a message.	No	The default value is <i>false</i> .
SupportsStatusPolling	For certain protocols, an active polling of the remote gateway must be performed to check the status of a message previously sent. This property indicates whether the driver supports such status polling. If set to <i>true</i> , the Messaging Engine invokes the driver connection's <code>getStatus()</code> operation.	No	The default value is <i>false</i> .
SupportsTracking	Supports Tracking operation on a message.	No	The default value is <i>false</i> .

38.2.2 Configuring the E-Mail Driver

The E-Mail Driver both sends and receives messages (that is, its *Capability* property is set to *BOTH* by default). The E-Mail Driver sends messages over SMTP and uses both IMAP and POP3 for receiving messages. In addition to the common properties described in [Table 38–1](#), the E-Mail Driver includes a set of specific properties (listed in [Table 38–2](#)). ([Table 38–2](#) also includes some of the common properties as they relate to the E-Mail Driver.)

Table 38–2 E-Mail Driver-Specific Properties

Name	Description	Mandatory?	Default Value
Capability	Sets the driver's capability to send or receive messages. The values are <i>SEND</i> , <i>RECEIVE</i> , and <i>BOTH</i> .	Yes	The default value is <i>BOTH</i> . This value set for this property affects the following: <ul style="list-style-type: none"> ■ <i>IncomingMailIDs</i> ■ <i>IncomingMailServer</i> ■ <i>IncomingMailServerPort</i> ■ <i>IncomingUserIDs</i> ■ <i>IncomingUserPasswords</i> ■ <i>MailAccessProtocol</i> ■ <i>OutgoingMailServer</i> ■ <i>OutgoingMailServerPort</i>
InstanceName	The name of the E-Mail Driver instance.	Yes	The default value is <i>Email-Driver</i> .
SupportedContentTypes	The content type supported by the E-Mail Driver instance.	Yes	The default content types include <i>text/plain</i> , <i>text/html</i> , <i>multipart/mixed</i> , <i>multipart/alternative</i> , <i>multipart/related</i> .
SupportedStatusTypes	The status types supported by the E-Mail Driver instance.	No	Default values include: <ul style="list-style-type: none"> ■ <i>DELIVERY_TO_GATEWEAY_SUCCESS</i> ■ <i>DELIVERY_TO_GATEWAY_FAILURE</i> ■ <i>USER_REPLY_ACKNOWLEDGEMENT_SUCCESS</i> ■ <i>USER_REPLY_ACKNOWLEDGEMENT_FAILURE</i>
AutoDelete	Enter <i>true</i> to enable the driver to mark messages as deleted once they have been processed; enter <i>false</i> to retain e-mails after they have been processed. Messages received using POP3 are always deleted after they have been processed.	No	The default value is <i>true</i> .
CheckMailFreq	The frequency, in seconds, in which the driver retrieves messages from the mail server.	No	The default value is 5 seconds.

Table 38–2 (Cont.) E-Mail Driver-Specific Properties

Name	Description	Mandatory?	Default Value
IncomingMailIDs	A comma-separated list of e-mail addresses that correspond to user names. Each e-mail address must occupy the same position in the list as their user name counterparts (listed in <i>IncomingUserIDs</i>).	No. Enter a value if the driver instance supports receiving only (that is, the driver instance's <i>Capability</i> property is set to <i>RECEIVE</i> or <i>BOTH</i>).	N/A
IncomingMailServer	The host name of the incoming mail server.	No. Enter a value if the driver instance supports receiving only (that is, the driver instance's <i>Capability</i> property is set to <i>RECEIVE</i> or <i>BOTH</i>).	N/A
IncomingMailServerPort	The port number of the IMAP or POP3 mail server. For example, enter a standard port number for the IMAP server, such as 143 or 993 (for S-IMAP, IMAP over SSL). Likewise, enter a standard port number for the IMAP server, such as 100 or 995 (S-POP, POP over SSL).	No	N/A
IncomingMailServerSSL	Enter true to enable the Messaging Enabler to communicate using SSL (Secure Sockets Layer). If this property is set to true, then you must set the <i>IncomingMailServerPort</i> to a port commonly used for SSL, such as 993. [[??]]	No	The default value is false (SSL is not used).
IncomingUserIDs	A comma-separated list of user names of the mail accounts from which the E-Mail Driver instance polls.	No. Enter a value if the driver instance supports receiving only (that is, the driver instance's <i>Capability</i> property is set to <i>RECEIVE</i> or <i>BOTH</i>).	N/A
IncomingUserPasswords	A comma-separated list of passwords that correspond to the user names. Each e-mail address must occupy the same position in the list as their user name counterparts (listed in <i>IncomingUserIDs</i>).	No	N/A

Table 38–2 (Cont.) E-Mail Driver-Specific Properties

Name	Description	Mandatory?	Default Value
MailAccessProtocol	The protocol used for receiving e-mail. The E-Mail driver uses either IMAP or POP3 to receive incoming e-mail.	No. Enter a value if the driver instance supports receiving (that is, the driver instance's <i>Capability</i> property is set to <i>RECEIVE</i> or <i>BOTH</i>).	IMAP
MailDelFreq	The frequency, in seconds, to purge processed e-mails. Enter a negative value, such as -1, to prevent the driver instance from purging processed messages. The driver instance always marks messages received over POP3 as deleted once they have been processed.	No	The default value is 600 seconds.
OutgoingDefaultFromAddr	The default <i>from</i> address of the e-mail (if none is provided in the message).	No	N/A
OutgoingMailServer	The name of the SMTP server.	No. Enter a value if the driver instance supports sending only (that is, the driver instance's <i>Capability</i> property is set to <i>SEND</i> or <i>BOTH</i>).	N/A
OutgoingMailServerPort	The port number of the outgoing SMTP server.	No	The default port number for the SMTP server is typically 25.
OutgoingMailServerTLS	If set to <i>true</i> , then the Messaging Enabler uses TLS (Transport Layer Security) encryption when communicating with the SMTP server.	No	N/A
OutgoingPassword	The password used for SMTP authentication.	No. Enter a password only if SMTP authentication is supported by the SMTP server.	N/A
OutgoingUsername	The user name used for SMTP authentication.	No. Enter a user name only if SMTP authentication is supported by the SMTP server.	N/A

Table 38–2 (Cont.) E-Mail Driver-Specific Properties

Name	Description	Mandatory?	Default Value
IncomingProcessingChunkSize	The maximum number of messages processed per message polling.	No	The default value for each chunk is 100.
ReceiveFolder	The name of the folder (such as INBOX) from which the driver polls messages.	No	INBOX
RetryLimit	The number of times that the driver attempts to reconnect when disconnected from the incoming mail server. Enter -1 for unlimited retries.	No	N/A

38.2.3 Configuring the SMPP Driver

SMPP (Short Message Peer-to-Peer) is one of the most popular GSM SMS protocols. Notification Service includes a pre-built implementation of the SMPP protocol as a driver that is capable of both sending and receiving short messages. If the sending feature is enabled, the SMPP driver opens one TCP connection to the SMS-C (Short Message Service Center) as a transmitter for sending. If the driver's receiving feature is enabled, it opens another connection to the SMS-C as a receiver for receiving. Only two TCP connections (both initiated by the driver) are needed for all communication between the driver and the SMS-C.

Note: The SMPP Driver driver implements Version 3.4 of the SMPP protocol and only supports connections to an SMS-C that supports this version.

In addition to the common properties described in [Table 38–1](#), the SMPP driver includes the following properties that are associated with configuring access to the remote gateway and protocol- and channel-specific behavior (described in [Table 38–3](#)).

Table 38–3 SMPP Driver-Specific Properties

Name	Description	Mandatory?	Default Value
InstanceName	The name of the SMPP driver instance.	Yes	The default value is <i>SMPP-Driver</i> .
SupportedContentTypes	The content type supported by the driver.	Yes	The default content type is <i>text/plain</i> .
SupportedStatusTypes	The status types supported by the SMPP driver.	No	The supported status types are: <ul style="list-style-type: none"> ▪ DELIVERY_TO_GATEWAY_SUCCESS ▪ DELIVERY_TO_GATEWAY_FAILURE
SmsAccountId	The Account Identifier on the SMS-C.	Yes	N/A

Table 38-3 (Cont.) SMPP Driver-Specific Properties

Name	Description	Mandatory?	Default Value
SmsServerHost	The name (or IP address) of the SMS-C server.	Yes	N/A
TransmitterSystemId	The account ID that is used to send out messages.	Yes	N/A
ReceiverSystemId	The account ID that is used to receive messages.	Yes	N/A
TransmitterSystemType	The type of transmitter system.	Yes	The default value is <i>Logica</i> .
ReceiverSystemType	The type of receiver system.	Yes	The default value is <i>Logica</i> .
TransmitterSystemPassword	The password of the transmitter system.	Yes	N/A
ReceiverSystemPassword	The password for the receiver system.	Yes	N/A
ServerTransmitterPort	The TCP port number of the transmitter system.	Yes	N/A
ServerReceiverPort	The TCP port number of the receiver system.	Yes	N/A
DefaultEncoding	The default encoding of the SMPP driver.	Yes	The default value is <i>UCS2</i> .
EncodingAutoDetect	If set to <i>true</i> (the default), the SMPP driver encodes automatically.	Yes	The default value is <i>true</i> .
LocalSendingPort	The local TCP port used by the SMPP driver to messages to the SMS-C.	Yes	N/A
LocalReceivingPort	The local TCP port used by the SMPP drivers to receive messages from the SMS-C.	Yes	N/A
LocalAddress	The host name (or IP address) of the server that hosts the SMPP driver.	No	N/A
WindowSize	The window size for SMS. This value must be a positive number.	No	The default value is 1.
EnquireInterval	The interval, in seconds, to send an enquire message to the SMS-C.	No	The default value is 30.
ThrottleDelay	The delay, in seconds, between throttles.	No	The default value is 15.
BindRetryDelay	The delay, in seconds, for a binding retry.	No	The default value is 30.
RegisteredDeliveryMask	The delay, in seconds, for a binding retry. [[?]]	No	The default value is 30.

Table 38–3 (Cont.) SMPP Driver-Specific Properties

Name	Description	Mandatory?	Default Value
RangeSetNull	Set to <i>true</i> to set the <i>address range</i> field of BIND_RECEIVER to <i>null</i> . Set to <i>false</i> (the default value) to set the address range field to <i>SmsSystemId</i> .	No	The default value is <i>false</i> .
PriorityAllowed	The highest priority allowed for the SMPP driver. The range is 0 (normal) to 3 (highest).	No	The default value is 0.
BulkSending	Setting this value to <i>true</i> (the default) enables sending messages in bulk to the SMS-C [???].	No.	The default value is <i>true</i> .
PayloadSending	If set to true, the SMPP driver always uses the message payload properties when sending messages to the SMS-C. [???]	No	The default value is false.
SourceTon	The Type of Number (TON) for ESME address(es) served through SMPP receiver session.	No	The default value is 0.
SourceNpi	The Numbering Plan Indicator (NPI) for ESME address(es) served through the SMPP receiver session.	No	The default value is 0.
DestinationTon	The Type of Number (TON) for destination.	No	The default value is 0.
DestinationNpi	The Numbering Plan Indicator (NPI) for destination.	No	The default value is 0.
ExtraErrorCode	A comma-separated list of error codes.	No	N/A
MaxChunks	The maximum SMS chunks for a message.	No	The default value is -1 (no maximum).
ChunkSize	The size of each SMS message chunk.	No	The default value is 160.
LongMessageSending	Supports sending long messages.	No	N/A
DatagramMessageMode	Supports Datagram Message mode.	No	N/A

38.2.4 Configuring the XMPP Driver

XMPP (Extensible Messaging and Presence Protocol) is an open, XML-based protocol for Instant Messaging and Presence. XMPP consists of a client-server architecture, one that resembles an e-mail network. XMPP servers are decentralized, enabling anyone to set up their own server. Messaging is achieved as in the e-mail network, where

recipients are addressed by a user name and a host name (for example: *username@hostname*). In the XMPP network, users are identified by an XMPP (Jabber) ID, which consists of a user name and the host name of the particular XMPP server to which the user connects. An end user of XMPP connects to an XMPP server using an XMPP client to send instant messages to other XMPP users. XMPP has an extensible and modular architecture. It integrates with proprietary IM networks such as Yahoo!, MSN, AOL and ICQ using transport gateways that can connect to these networks. XMPP, however, is not the only protocol network available for instant messaging.

The XMPP Driver provides end users with unidirectional and bidirectional access from Oracle Application Server for real-time instant messaging (IM) through XMPP. This driver enables end users to receive alert notifications or chat through their IM client of choice.

Required Components for the XMPP Driver

To use the XMPP Driver in OracleAS Notification Service, you must first have access to a Jabber/XMPP server and an XMPP account to enable the OracleAS Notification Service XMPP Driver instance to login. Because the XMPP Driver includes configuration properties that enable the Oracle Messaging Enabler to communicate with users through such proprietary IM networks as Yahoo!, MSN, AOL and ICQ, you must create accounts on the networks that connect to XMPP Driver to enable communication between the users and the Messaging Enabler. **[[Are there recommended versions for the chat clients?]]**

The XMPP Driver uses or requires the following third-party software

Table 38–4 Requirements for the XMPP Driver

Software Component	Instructions	Version(s)
JabberBeans	This driver uses the JabberBeans Java library to connect to a Jabber/XMPP Instant Messaging Server. This driver includes a licensed copy of JabberBeans (version 0.9.1).	0.9.1
XMPP Server	To download and install your own Jabber/XMPP server, pick and install a server from http://www.jabber.org . Note: You do not need to install your own XMPP Server if you have access to an existing server. For a list of public servers, see http://www.jabber.org .	N/A
Yahoo!, MSN, AOL (AIM), and ICQ Transport Gateways	Refer to the Jabber/XMPP server's transport installation guide for information on installing and configuring one or more transports to connect to proprietary IM gateways. The transport gateway is an optional component.	N/A

In addition to the properties described in [Table 38–1](#), the XMPP driver also includes the following properties that relate to configuring access to the remote gateway and protocol- or channel-specific behavior (described in [Table 38–5](#)).

Table 38–5 XMPP-Specific Driver Properties

Name	Description	Mandatory?	Default
IMServerHost	The host name of the Jabber server. For multiple servers, use a comma-separated list (for example, enter <i>my1.host.com, my2.host.com</i>). If only one host name is specified, it will be used for all accounts.	Yes	N/A
IMServerPort	The corresponding comma-separated list of Jabber server ports (for example, enter <i>5222, 5222</i>)	Yes	The default value is <i>5222</i> .
IMServerUsername	A list of Jabber user names for login. These user accounts will be automatically created, if necessary, on the corresponding Jabber servers. If you have multiple servers listed in the <i>IMServerHost</i> and <i>IMServerPort</i> properties, then there must be an equal number of user names (one user name per server). If you have only one server listed in these properties, then all of the user names listed in this property will use that server. For example, enter <i>oracleagent1, oracleagent2</i> . You can also enter a complete Jabber ID if its domain name is different from the Jabber server host name (for example, enter <i>oracleagent1@host.com</i>).	Yes	N/A
IMServerPassword	A comma-separated list of passwords for each user name listed for the <i>IMServerUsername</i> property.	Yes	N/A
HTTPUseProxy	Use HTTP Proxy.	No	The default value is <i>false</i> .
HTTPNonProxyHosts	A comma-separated list of non-proxy hosts.	No	N/A
HTTPProxyHost	The name of the HTTP proxy host.	No	N/A
HTTPProxyHost	The port of the HTTP proxy.	No	N/A
YahooEnable	Enable or Disables Yahoo! Transport for each user account specified in the <i>IMServerUserName</i> property in a comma-separated list. Set to <i>true</i> to enable; set to <i>false</i> to disable.	No	N/A
YahooUsername	A comma-separated list of Yahoo! account IDs for each user account listed in the <i>IMServerUserName</i> property. Entering valid Yahoo! account information enables Yahoo! users to access OracleAS Wireless applications through instant messaging. The Yahoo! account IDs must also be registered on Yahoo!. Leave the entries blank for accounts without Yahoo!.	No	N/A
YahooPassword	A comma-separated list of Yahoo! account passwords that correspond to the IDs entered in the <i>YahooUsername</i> property.	No	N/A
MSNEnable	Enables or disables MSN Transport for each user listed in the <i>IMServerUsername</i> property; set to <i>true</i> to enable; set to <i>false</i> to disable.	No	N/A
MSNUsername	A comma-separated list of MSN Messenger (also known as .NET passport) account IDs for each user account listed in the <i>IMServerUsername</i> property. Leave entries blank for accounts without MSN. Entering valid .NET account information enables MSN Messenger users to access OracleAS Wireless applications through instant messaging. The MSN account IDs must also be registered as .NET passports.	No	N/A

Table 38–5 (Cont.) XMPP-Specific Driver Properties

Name	Description	Mandatory?	Default
MSNPASSWORD	A comma-separated list that corresponds to the accounts listed in the <i>MSNUsername</i> property.	No	N/A
AOLEnable	Enables or disables AOL IM (AIM) Transport for each user account specified in the <i>IMServerUserName</i> property's comma-separated list. Set to <i>true</i> to enable; leave this property blank or set to <i>false</i> to disable.	No	N/A
AOLUsername	A comma-separated list of AOL IM (AIM) account IDs for each user account in the <i>IMServerUserName</i> property. Entering valid AOL account information enables AOL users to access OracleAS Wireless applications through instant messaging. The account IDs must be registered with AOL. Leave entries blank for accounts without AOL.	No	N/A
AOLPassword	A comma-separated list of AOL IM account passwords that correspond to the accounts listed in the <i>AOLUsername</i> property.	No	N/A
ICQEnable	Enables or disables ICQ IM Transport for each user account specified above in a comma-separated list. Set to <i>true</i> to enable; leave blank or set to <i>false</i> to disable.	No	N/A
ICQUsername	A comma-separated list of ICQ account IDs for each user account listed in the <i>IMServerUserName</i> property. Entering valid ICQ account information enables ICQ users to access OracleAS Wireless applications through instant messaging. The ICQ accounts must be registered with ICQ. Leave entries blank for accounts without ICQ.	No	N/A
ICQPassword	A list of ICQ account passwords that correspond to the user IDs entered in the <i>ICQUsername</i> property.	No	N/A
RetryLimit	The number of times the driver attempts to reconnect to the Jabber server. Enter <i>-1</i> for unlimited retries.	No	N/A
RetryInterval	The interval, in seconds, between the driver's attempts to reconnect to the Jabber server.	No	N/A

38.2.5 Configuring the VoiceGenie Driver

The VoiceGenie Driver supports VoiceGenie's outbound call protocol. In addition to the common properties described in [Table 38–1](#), the VoiceGenie Driver includes a set of specific properties (listed in [Table 38–6](#)). ([Table 38–6](#) also includes some of the common properties as they relate to the VoiceGenie Driver.)

Table 38–6 VoiceGenie-Specific Drivers

Name	Description	Mandatory?	Default Value
InstanceName	The name of the instance (for internal use only).	Yes	VoiceGenieDriver
Capability	Sets the driver's capability to send or receive messages. The values are <i>SEND</i> , <i>RECEIVE</i> , and <i>BOTH</i> .	Yes -- The VoiceGenie driver supports only sending.	<i>SEND</i>
SendingQueuesInfo	The information for the Driver Sending Queue.	Yes	The default value is: <i>OracleNS/QueueConnectionFactory:OracleNS/Queus/OracleSDPMDriverDefSndQ1</i> [[??]]
SupportedDelieveryTypes	The delivery type supported by the VoiceGenie driver.	Yes	<i>VOICE</i>
SupportedContentTypes	The content type supported by the VoiceGenie driver.	Yes	The content types supported by the VoiceGenie driver are: <ul style="list-style-type: none"> ■ text/vxml ■ text/v+xml
SupportedStatusTypes	The status types supported by the VoiceGenie driver.	No	The status types support by the VoiceGenie driver are: <ul style="list-style-type: none"> ■ DELIVERY_TO_GATEWAY_SUCCESS ■ DELIVERY_TO_GATEWAY_FAILURE
VoiceGenieOutboundServletURI	The URL of the VoiceGenie gateway.	Yes	N/A
VoiceGenieOutboundServletUserName	The user name of the VoiceGenie gateway.	No	N/A
VoiceGenieOutboundServletPassword	The password of the VoiceGenie gateway.	No	N/A
VoiceGenieOutboundServletDNIS	The number that appears in the recipient's ID display.	No	N/A
VoiceGenieReceiveURL	The URL of the servlet which handles incoming requests from the VoiceGenie VoiceXML Gateway.	Yes	N/A

38.2.6 Configuring the Proxy Driver

The Proxy Driver acts as a Messaging Web Service client to a Fusion Middleware Messaging server hosted elsewhere in the intranet or Internet. It uses SOAP over HTTP (the 11g Fusion Middleware Messaging Web Service protocol) to send messages and receive messages as well as return message delivery status. The Proxy Driver relays messages from one UMS instance to another. It can be used to relay traffic from multiple instances in an Intranet to a terminating instance that has all of the protocol-specific drivers configured to an external gateway such as an SMSC, or to an

SMTP or IMAP mail server. [Table 38–7](#) lists the parameters specific to the Proxy Driver as well as those described in [Section 38.2.1](#) as they relate to the Proxy Driver.

Table 38–7 Parameters of the Proxy Driver

Name	Description	Mandatory?	Default Value
Instance Name	The name of the Proxy Driver instance.	Yes	Proxy-Driver
Capability	Sets the driver’s capability to send or receive messages. The values are <i>SEND</i> , <i>RECEIVE</i> , and <i>BOTH</i> .	Yes	The default value is <i>BOTH</i> .
SupportedDeliveryTypes	The delivery types supported by the driver.	Yes	The default values are: <ul style="list-style-type: none"> ■ SMS ■ IM ■ Voice ■ E-Mail ■ Worklist
SupportedContentTypes	The content type supported by the driver.	Yes	The default value is an asterisk (*), as the Proxy Driver supports all content types.
SupportedStatusTypes	The status types supported by the driver.	No	Default values include: <ul style="list-style-type: none"> ■ DELIVERY_TO_GATEWAY_SUCCESS ■ DELIVERY_TO_GATEWAY_FAILURE
GatewayURL	The URL to the hosted 11g UMS Web Service gateway. The URL is in the following format: http://<host>:<port>/sdpmessaging/webservice	Yes	N/A
Password	The password of the username	No	N/A
CallbackWsURL	The URL that receives the incoming messages and statuses from the 11g UMS Web Service gateway. The URL is in the following format: http://<host>:<port>/sdpmessaging/driver-proxy/receive	No	N/A
CallbackWsVersion	The version number of receiving Web Service.	No	11.1.1.0.0

38.3 Deploying Drivers

Oracle Enterprise Manager 11g provides a wizard that steps you through deploying, undeploying, and redploying drivers.

Note: Oracle User Messaging Service must be deployed to an OC4J instance before you can deploy the driver EAR files. [\[\[?\]\]](#)

To access the deployment wizard, first select the *Farm* menu’s **J2EE Application Deployment** option and then select **Deploy** ([Figure 38–8](#)).

Figure 38–8 Accessing the Deployment Wizard

The **Deploy** option invokes the deployment wizard, which guides you through the deployment process through the following pages:

- The *Select Archive* page (Figure 38–9) is the first page of the wizard. To complete this page, point OC4J to location of the driver’s EAR (Enterprise Archive) file, which is typically located at `ORACLE_HOME/archives/applications`. This page also enables you to select the option to create or apply a deployment plan, a client-side aggregation of all of the configuration data needed to deploy an archive into OC4J. If you use an existing deployment plan, you enter its location. If you opt for new deployment plan, select the **Automatically Create a New Deployment Plan** option.

Tip: The wizard automatically creates a new deployment plan if you do not enter the location of an existing plan.

Figure 38–9 Deploying an Application: Entering the Archive Location

ORACLE Enterprise Manager 11g Grid Control

Select Archive Select Target Application Attributes Deployment Settings

Select Archive Help Cancel Step 1 of 4 Next

Specify the archive and deployment plan for the application to be deployed.

Archive

Oracle archive, SOA Application archive, Standard Java EE archive, Web Modules (WAR files), EJB Modules (EJB JAR files) and Resource Ada Modules (RAR files) can be deployed. You can also deploy an exploded archive that is present on the server where Enterprise Manager is run and accessible to Oracle Middleware Administration Server.

Archive is present on local host. Upload the archive to the server where Enterprise Manager is running.

C:\archives\applications\msgdriver-smpp.ear Browse...

Archive is already present on the server where Enterprise Manager is running.

Use the archive from a previously deployed application.

Browse Deployed Applications

Deployment Plan

The deployment plan is an file that contains the deployment settings for an application. If you do not have a deployment plan, one will be created automatically during the deployment process. Later in the deployment process, you can optionally edit the deployment plan and save it for a future deployment of this application.

Automatically create a new deployment plan.

Deployment plan is present on local host. Upload the deployment plan to the server where Enterprise Manager is running.

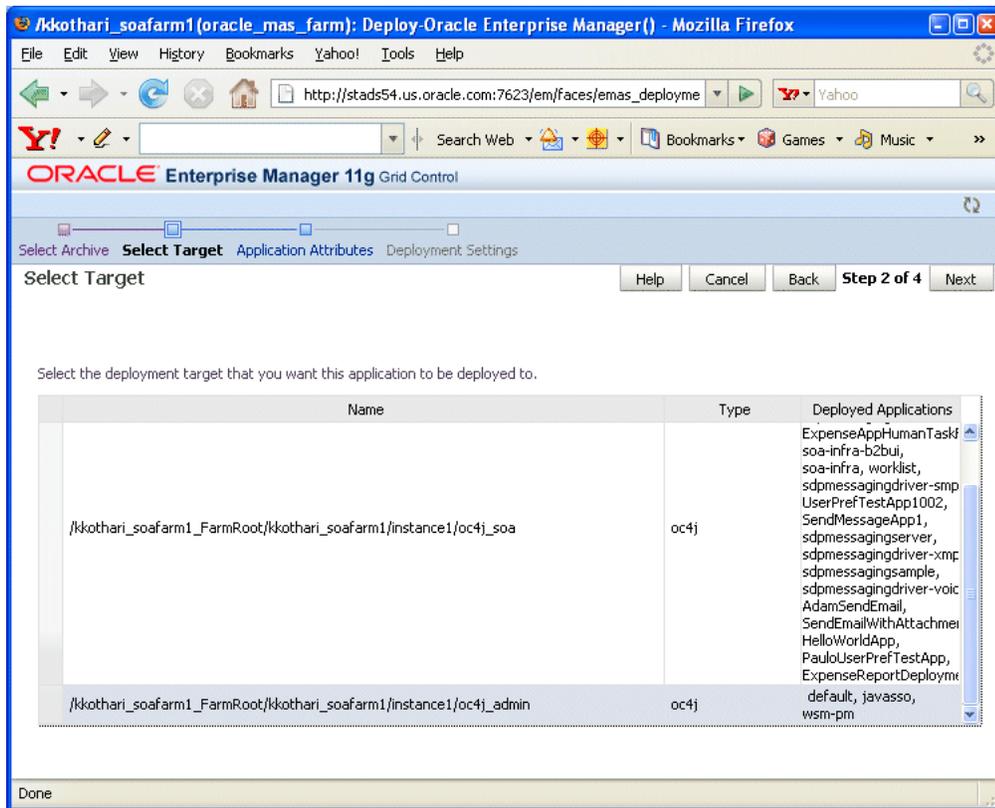
Browse...

Deployment plan is already present on server where Enterprise Manager is running.

- Clicking **Next** invokes the *Select Target* page (Figure 38–10). This page enables you to select the driver’s parent application.

Note: In general, the driver is a child of the *default* application. Do not deploy drivers as children of the Oracle User Messaging Service application. [??]

Figure 38–10 The Select Target Page



- The *Deployment Settings* page ([**graphic to be added**]) provides a tasks that enable you to edit the deployment plan.

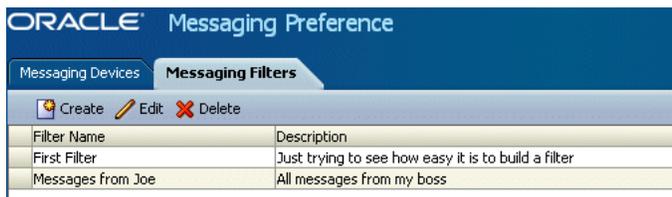
Complete deployment tasks as needed and then click **Deploy**. The confirmation page appears ([**graphic to be added**]).

The driver appears as an active application in the *Associated Drivers* pane ([Figure 38–5](#)).

38.4 Configuring Messaging Preferences

Workspace users route notifications to their devices using the User Messaging Preferences interface. This application, one accessed from the WebCenter Worklist, contains two tabs: the *Devices* tab ([Figure 38–3](#)), which enables users to register the e-mail clients and mobile devices on which they receive notifications and the *Messaging Filters* tab ([Figure 38–11](#)), where users specify the content they wish to receive and the device channels for receiving them.

Figure 38–11 The Messaging Filters Tab



38.4.1 Managing Devices

Any device that a user creates is associated with that user's system ID. In Oracle User Messaging Service, devices represent both physical devices, such as mobile phones, and also e-mail client applications running on desktops. The *Devices* tab enables users to perform the following tasks:

38.4.1.1 Creating a Device

To create a device:

1. Click **Create** (Figure 38–12).

Figure 38–12 The Create Icon



2. Enter a name for the device in the *Device Name* field.
3. A physical device, such as a PDA, can have one or several channels depending on its messaging capabilities. As illustrated in Figure 38–13, a device called *MyWorkPDA* can be the destination device for Voice, IM, SMS, and E-Mail channels.

Figure 38–13 Multiple Channels for a Single Device

Channel	Address	Action
VOICE	3105555555	✖
SMS	3105555555	✖
IM	scot.tiger@oracle.com	✖

Define the channels for the device by first selecting the transport type from the *Add Channels* list and then by entering the appropriate number or address.

4. Click **Add** (Figure 38–14).

Figure 38–14 The Add Icon



Click **Delete** (Figure 38–15) to remove a channel from a device.

Figure 38–15 The Delete Icon



5. Click **OK** to complete the device. The device appears on the *Devices* page, represented as subfolder, the contents of which are the device's channels. The *Devices* page enables you to edit or delete the device as well as set a device channel as the default destination for receiving notifications. Clicking **Cancel** discards the device.

38.4.1.2 Editing a Device

To edit a device, click **Edit** adjacent to the device folder. The editing page appears for the device, which enables you to add or change the device properties described in [Section 38.4.1.1, "Creating a Device"](#).

38.4.1.3 Deleting a Device

To delete a device, click **Delete** adjacent to the device folder.

38.4.1.4 Setting a Default Device

E-Mail is the default for receiving notifications. To set another device as the default, expand the device folder, and then click **Set as Default** adjacent to the selected device channel. A check mark ([Figure 38–16](#)) appears next to the selected device channel, designating it as the default means of receiving notifications.

Figure 38–16 *The Default Icon*



38.4.2 Creating Contact Rules using Filters

The *Messaging Filters* tab enables users to build filters that specify not only the type of notifications they wish to receive, but also the device 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 device channels, block device channels from receiving notifications, or send notifications to the first available device.

[Figure 38–17](#) illustrates the creation of a filter called *Messages from Joe*, written by an employee named Scott that retrieves all high priority messages addressed to him from his boss, Joe. Notifications that match all of the filter conditions are first directed to *(MyPDA)E-Mail* channel. Should this channel become unavailable, Oracle User Messaging Service transmits the notifications as text-to-speech (TTS) voice mails since Scott selected a voice channel called *(Mynewdevice)Voice* as the next available channel.

Figure 38–17 Creating a Filter

ORACLE Messaging Preference

Messaging Devices | **Messaging Filters**

Filter Name: Messages from Joe
 Description: Retrieve Important Messages from my boss

Condition
 Matching: All of the following conditions Any of the following conditions

Add Filter Condition: To isEqual scott.tiger@oracle.com +

Attribute	Operator	Value	Value2 (if required)	Delete
Priority	isEqual	High		✖
From	isEqual	joe@oracle.com		✖
To	isEqual	scott.tiger@oracle.com		✖

Action
 Messaging Option: Send to one of the selected channels (fail-over in the order)

Add Notification Channel: (MyDevice)IM +

Device and Channel	Channel Address	Up	Down	Delete
(MyPDA)EMAIL	650555555@carrier.com	↑	↓	✖
(Mynewdevice)VOICE	16505068691	↑	↓	✖

Cancel OK

38.4.2.1 Creating Filters

To create a filter:

1. Click **Create** (Figure 38–12). The *Create Filter* page appears (Figure 38–17).
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 38–14) 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 38–15).
9. Select one of the following delivery rules:
 - **Broadcast to All Channels** -- Select this option to send messages to every listed channel.
 - **Send to One of the Selected Channels (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.
 - **Do Not Send Any Message** -- Select this option to block the receipt of any messages that meet the filter conditions to all of the listed channels.
10. To set the delivery channels, select a channel from the *Add Notification Channel* list and then click **Add** (Figure 38–14). To delete a channel, click **Delete** (Figure 38–15).
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 One of the Selected Channels (Failover in the Order)*.
12. Click **OK** to complete the filter. Clicking **Cancel** discards the filter.

38.4.2.2 Editing a Filter

To edit a filter, first select it and then click **Edit** (Figure 38–6). The editing page appears for the filter, which enables you to add or change the device properties described in Section 38.4.2.1, "Creating Filters".

38.4.2.3 Deleting a Filter

To delete a filter, first select it and then click **Delete** (Figure 38–15).

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. It contains the following sections:

- [Section 39.1, "Overview of Parlay X Messaging Operations"](#)
- [Section 39.2, "Send Message Interface"](#)
- [Section 39.3, "Receive Message Interface"](#)
- [Section 39.4, "Oracle Extension to Parlay X Messaging"](#)
- [Section 39.5, "Parlay X Messaging Client API and Client Proxy Packages"](#)

Note: Oracle User Messaging Service also ships with a Java API implementation of the Parlay X API.

39.1 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 tables containing input/output message descriptions 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.

39.2 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 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.

39.2.1 sendMessage Operation

Table 39–1 describes message descriptions for the `sendMessageRequest` input in the `sendMessage` operation.

Table 39–1 *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 <code>senderAddress</code> , even if a separate <code>senderAddress</code> 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.
charging	common:ChargingInformation	Yes	Charging to apply to this message.
receiptRequest	common:SimpleReference	Yes	Defines the application endpoint, <code>interfaceName</code> and <code>correlator</code> that will be used to notify the application when the message has been delivered to a terminal or if delivery is impossible.

[Table 39–2](#) describes `sendMessageResponse` output messages for the `sendMessage` operation.

Table 39–2 *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.

39.2.2 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 39–3](#) describes the `getMessageDeliveryStatusRequest` input messages for the `getMessageDeliveryStatus` operation.

Table 39–3 *getMessageDeliveryStatusRequest Input Message Descriptions*

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	Identifier related to the delivery status request.

[Table 39–4](#) describes the `getMessageDeliveryStatusResponse` output messages for the `getMessageDeliveryStatus` operation.

Table 39–4 *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.

39.3 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.

39.3.1 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 39–5](#) describes the `getReceivedMessagesRequest` input messages for the `getReceivedMessages` operation.

Table 39–5 *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 39–6](#) describes the `getReceivedMessagesResponse` output messages for the `getReceivedMessages` operation.

Table 39–6 *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.

Table 39–6 (Cont.) *getReceivedMessagesResponse* Output Message Descriptions

Part Name	Part Type	Optional	Description
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.

39.3.2 *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 39–7 describes the *getMessageRequest* input messages for the *getMessage* operation.

Table 39–7 *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.

39.3.3 *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 39–8 describes the *getMessageURIsRequest* input messages for the *getMessageURIs* operation.

Table 39–8 *getMessageURIsRequest Input Message Descriptions*

Part Name	Part Type	Optional	Description
messageRefIdentifier	xsd:string	No	The identity of the message to retrieve.

Table 39–9 describes the `getMessageURIsResponse` output messages for the `getMessageURIs` operation.

Table 39–9 *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.

39.4 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.

39.4.1 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).

39.4.1.1 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 whitespace-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 `PolicyError` 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 39–10](#) describes the `startReceiveMessageRequest` input messages for the `startReceiveMessage` operation.

Table 39–10 *startReceiveMessageRequest Input Message Descriptions*

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	A registration identifier.
messageService ActivationNumber	xsd:anyURI	No	Message Service Activation Number.
criteria	xsd:string	Yes	Descriptive string.

There are no `startReceiveMessageResponse` output messages for the `startReceiveMessage` operation.

39.4.1.2 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 `PolicyError` will result.

[Table 39–11](#) describes the `stopReceiveMessageRequest` input messages for the `stopReceiveMessage` operation.

Table 39–11 *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.

39.5 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 via client classes in package `oracle.sdp.messaging.parlayx.v2_1.mm.service`:

- `SendMessageClient`

- `ReceiveMessageClient`
- `ReceiveMessageManagerClient`

Each client class allows a client application to invoke the operations in the corresponding interface. These classes also provide generic methods to set standard Web Service parameters such as the remote gateway URL (using the `setEndpoint()` method), and the remote security credentials (using the `setUsername()` and `setPassword()` methods). 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 (e.g. `SendMessageClient.sendMessage()`, etc). For examples of how to use this API, see the Messaging samples, and specifically `sdpmessagingsample-parlayx-src.zip`.

User Messaging Preferences

This chapter describes the User Messaging Preferences that are packaged with Oracle User Messaging Service. It describes how the user configures User Messaging Preferences and how filters work. This chapter contains the following sections:

- [Section 40.1, "Introduction"](#)
- [Section 40.2, "Configuring Filters and Conditions"](#)
- [Section 40.3, "Configuring Device Destination Addresses"](#)

40.1 Introduction

User Messaging Preferences allows a user who has access to multiple devices or channels (delivery types) to deliver notifications to a preferred device using filters. These filters are also known as notification delivery preferences. An Oracle database, a Toplink-compliant database, or a file-based database stores device addresses, rule sets, and user devices.

40.1.1 Terminology

User Messaging Preferences defines the following terminology:

- **Device:** a physical device, such as a phone, or PDA.
- **Device address:** one of the addresses that a device 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, e-mail, 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 Device*.

40.1.2 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 device 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 an online auction application. For example, user *Alex* is interested in being notified through SMS and IM messages for bids of an item (for example, source = 1234) on eBay that are less than 200. Alex's preferences can be stated as follows:

Example 40–1 Notification Delivery Preferences

Rule (1): if (application = eBay) AND (source = 1234) AND (amount < 200) then notify [Alex] using SMS and IM

Rule (2): if (application = eBay) AND (source = 1234) AND (amount >= 200) then do not notify [Alex]

A runtime service, the Oracle Rules Engine, evaluates the filters to process the notification delivery of user requests.

40.1.3 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.

40.1.3.1 Data Types

Table 40–2 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 40–1 Data Types Supported by User Messaging Preferences

Data Type	Comparison Operators
Date	<, >, between, <=, >=
Number	<, >, between, <=, >=
String	==, !=, contains, not contains

Note: The String data type does not support regular expressions.

System Terms

Table 40–2 lists system terms, which are pre-defined business terms. Administrators cannot extend the system terms.

Table 40–2 System Terms Supported by User Messaging Preferences

System Term	Data Type
Date	Date
Time	Number (0-23)

40.1.3.2 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 Developer'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 40–3 lists the pre-defined business terms supported by User Messaging Preferences.

Table 40–3 Pre-defined Business Terms for User Messaging Preferences

Business Term	Data Type
Organization	String
Customer Name	String
Customer Type	String
Status	String
Amount	Number
Due Date	Date
Process Type	String
Expense Type	String
Total Cost	Number
Processing Time	Number
Order Type	String
Service Request Type	String
Group Name	String
Source	String
Classification	String
Duration	Number
User	String
Role	String

40.1.4 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 devices in the broadcast address list.
- **Failover:** Send a message serially to devices in the address list until one successful message is sent. This means performing a send to the next device when the current device 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 Device:** Do not send a message to any device.

Tip: User Messaging Preferences does not provide a filter action that instructs "do not send to a specified device." 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 Devices page in Enterprise Manager.

40.2 Configuring Filters and Conditions

A user can specify multiple one or more conditions using system terms and business terms with the appropriate comparison operators. A user can have only one filter, which can be composed on one of more conditions.

40.2.1 Sample Conditions

User Messaging Preferences supports limited types of delivery preference rules. The following are some sample conditions that a User Messaging Preferences user can create:

1. **If (term is "Application" and the value is "Expense") AND (term is "Amount" and the value is greater than 1000) then** send to all of the following devices.
2. **If (term is "Date" and the value is between "Jun 17, 2006" and "Jun 26, 2006) then** send to the following address.
3. **If (term is "Time" and the value is between "12:00" and "13:00") AND (term is "Status" and the value is "success") then** send to one of the following devices.

40.2.2 Rule Activation Flow

User Messaging Preferences applies the following criteria to select conditions for activation:

- There is no priority associated with each conditions. Therefore, within a set of activation of rules, the first condition in the list will be executed first.
- The User Messaging Preferences user can access the User Messaging Preferences UI through the SOA Worklist application and prioritize the rules by rearranging their physical positions.
- If no condition is met, the service will return the user default device address as configured in the Messaging Devices page in Enterprise Manager.
- If the action of the filter is "Do Not Send to Any Device", an empty list will be returned.

- All filter sessions are stateless. User Messaging Preferences has no knowledge of previous facts or activation results.
 - There is no decision-branching between two rules.
 - There can be no aggregation rules such as number of notifications sent to a device address in a time period.
 - There can be no cross-user (group) rules.

For a given rule, the relationship between all comparisons can be either "match all conditions" or "match any condition". This precludes the specification of a mixture of ANDs and ORs.

40.3 Configuring Device Destination Addresses

A User Device is a device associated with a User Messaging Preferences user, and can include a user ID, a name, a set of device addresses, and a description.

The device address represents a communication address in a User Device (for example, a telephone number or e-mail address). A User Device can contain multiple addresses (such as addresses representing a telephone number, an e-mail address, and an instant messaging client) and therefore can contain multiple device addresses.

Device addresses have the following properties:

- address: String
- deliveryType: DeliveryType (from the Messaging API)
- encoding: String
- device: UserDevice
- createdDate: Date
- lastModifiedDate: Date
- version: int
- valid: boolean

`DeliveryType` enumerates all available delivery type in Oracle User Messaging Service. These constants are available:

- EMAIL
- EMS
- FAX
- IM
- MMS
- MOBILE_PUSH
- ONE_WAY_PAGER
- SMS
- TWO_WAY_PAGER
- VOICE

Send Message to User Specified Channel Sample Application

This chapter describes how to build and run the Send Message to User Specified Channel sample application provided with Oracle User Messaging Service. This chapter contains the following sections:

- [Section 41.1, "Overview"](#)
- [Section 41.2, "Installing and Configuring SOA and User Messaging Service"](#)
- [Section 41.3, "Building the Sample"](#)
- [Section 41.4, "Creating a Deployment Profile"](#)
- [Section 41.5, "Creating a New Application Server Connection"](#)
- [Section 41.6, "Deploying the Application"](#)
- [Section 41.7, "Configuring Drivers"](#)
- [Section 41.8, "Configuring User Messaging Preferences"](#)
- [Section 41.9, "Testing the Sample"](#)

41.1 Overview

The “Send Message to User Specified Channel” sample 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.

41.1.1 Provided Files

The following files are included in the sample application:

- `SendMessage.pdf` – this document.
- `Project` – the directory containing Oracle JDeveloper project files.
- `Readme.txt`.
- Release notes

41.2 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:

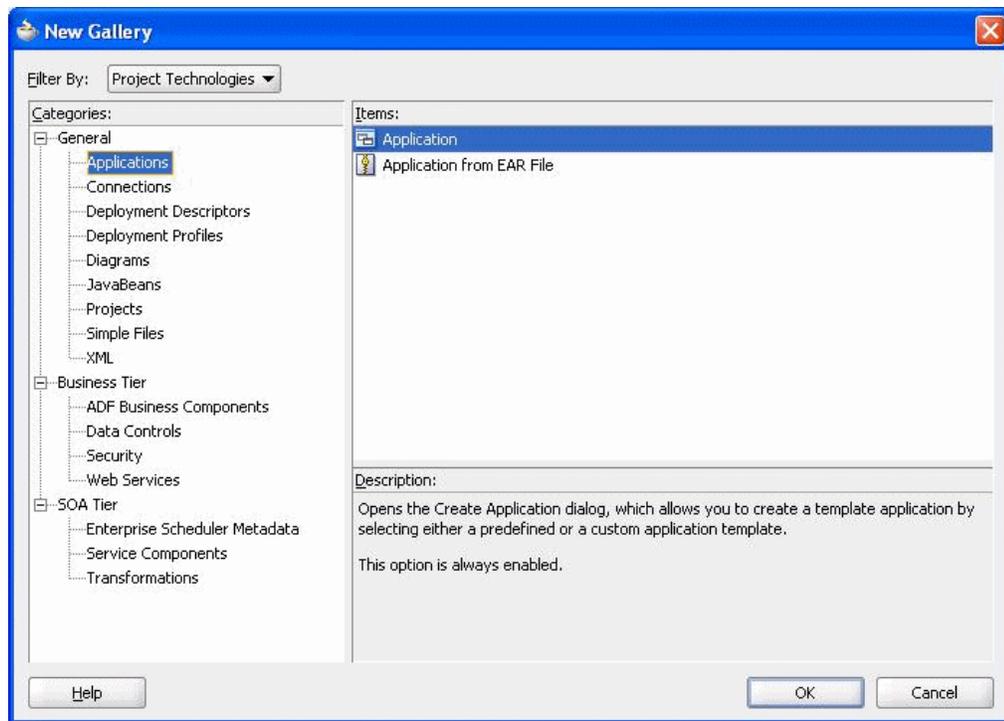
1. Configure AS11 soa-infra to access an e-mail server for uploading e-mails.
2. Configure additional drivers if required as described in "Configuring Drivers".
3. Restart the server.

41.3 Building the Sample

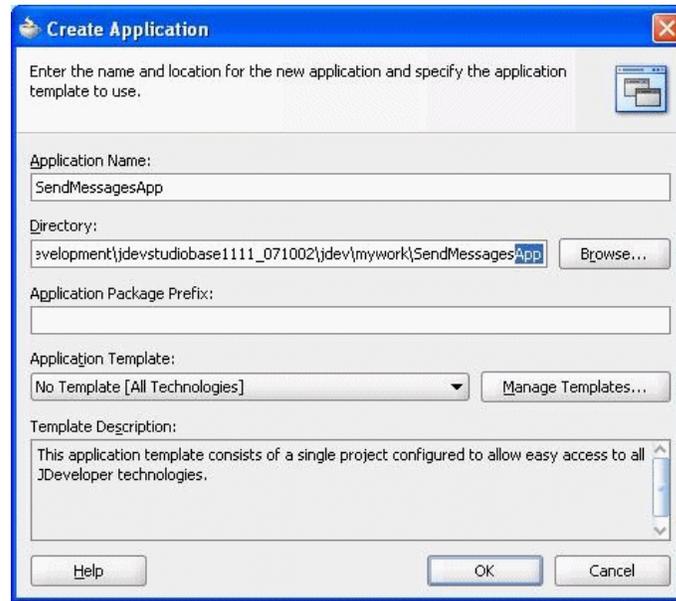
Performing the following procedure of building the sample from scratch allows you to learn how to add messaging to your SOA Composite Applications, and use User Messaging Preferences.

1. Open Oracle JDeveloper 11g.
2. Create a new application by selecting **New, General, Applications**, and **Application** (Figure 41-1). Click OK.

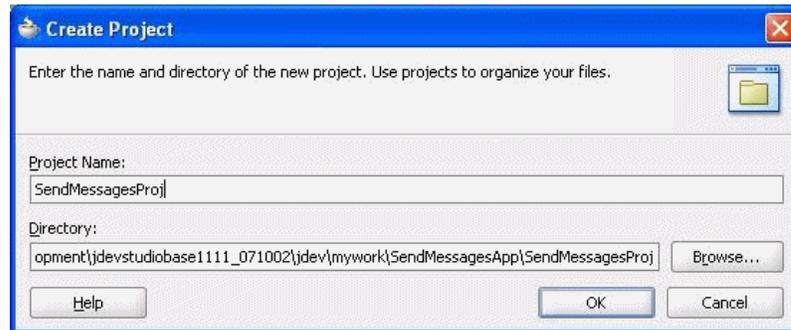
Figure 41-1 Creating a New Application (1 of 2)



3. Enter the *Application Name* and click OK (Figure 41-2).

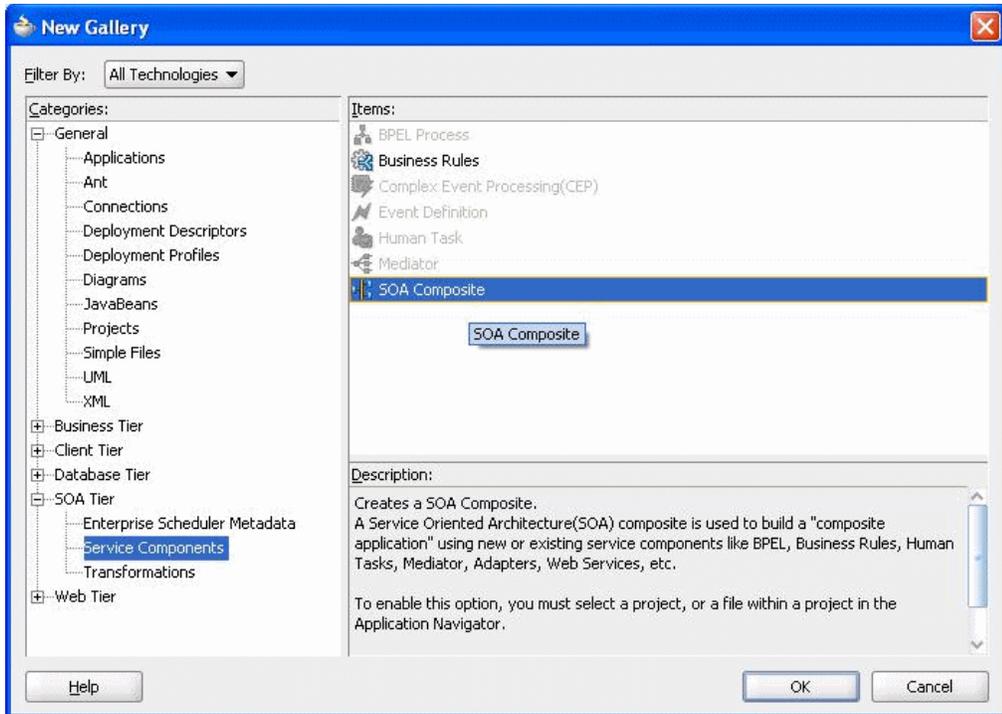
Figure 41–2 Creating a New Application (2 of 2)

4. Enter the name for the project and click **OK** (Figure 41–3).

Figure 41–3 Creating a New Project

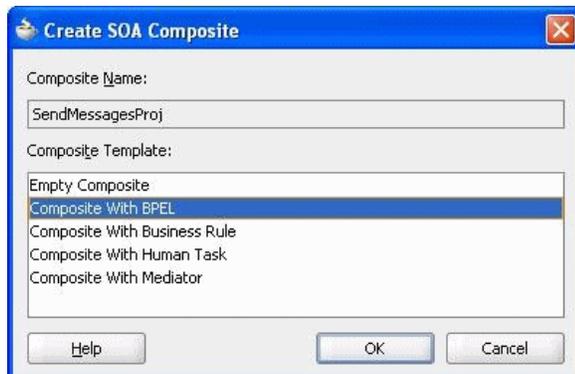
5. Select *SendMessageProj*, right click and select **New, SOA Tier, Service Components, and SOA Composite** (Figure 41–4).

Figure 41–4 Creating the SOA Composite



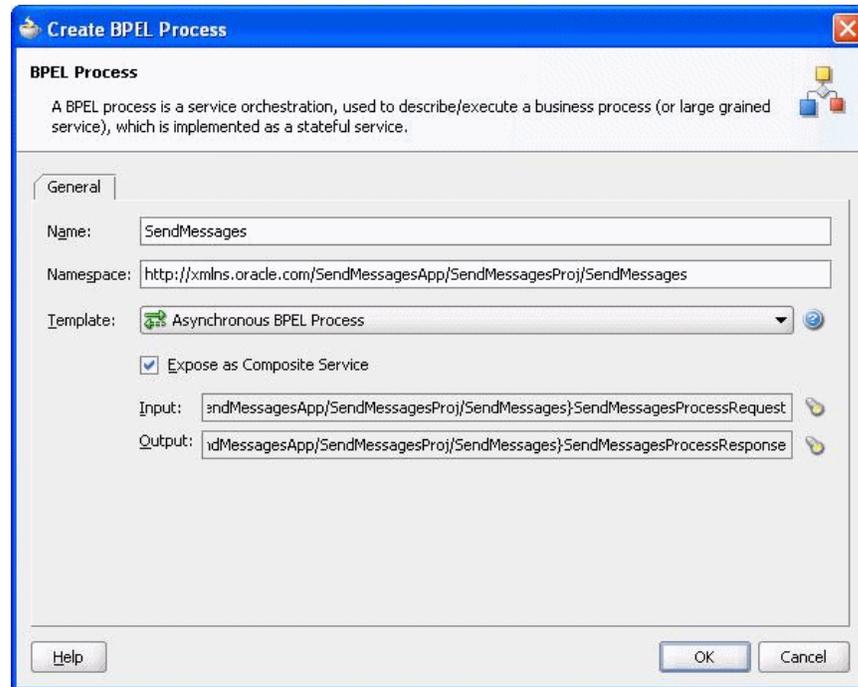
6. Select the *Composite With BPEL* composite template. Click **OK**. (Figure 41–5).

Figure 41–5 Creating the SOA Composite



7. In the *Create BPEL Process* window, enter the BPEL process name as “SendMessage.” Click **OK**.

Figure 41–6 Defining the BPEL Process



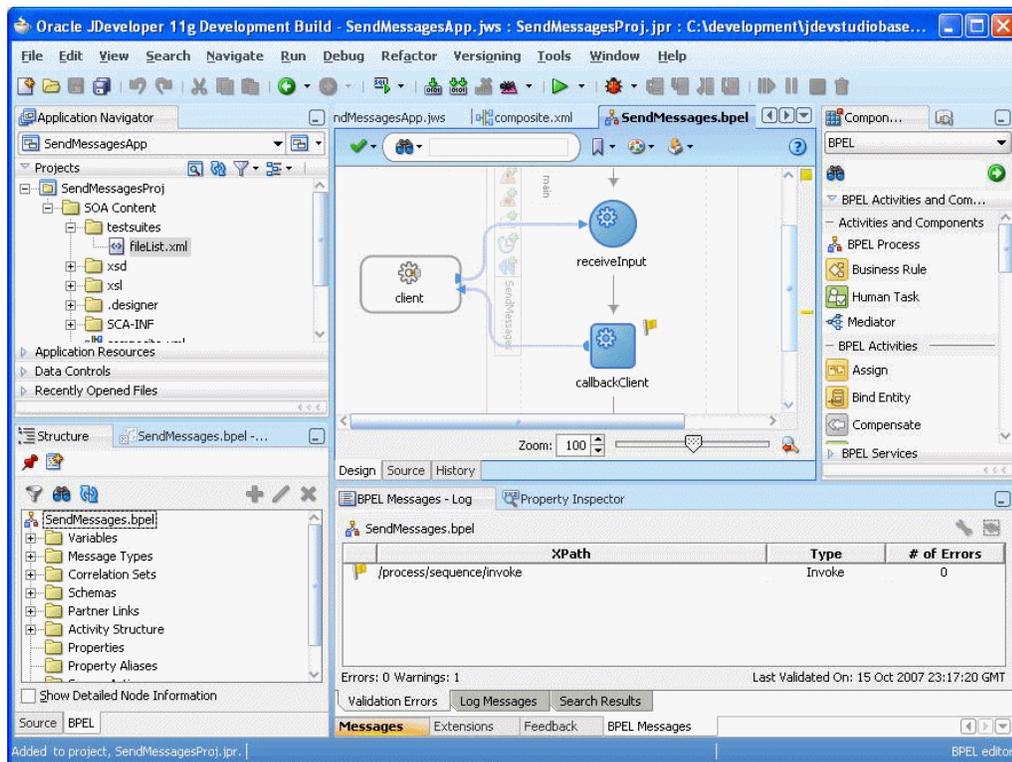
You have now created an empty and default BPEL application (Figure 41–7).

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.

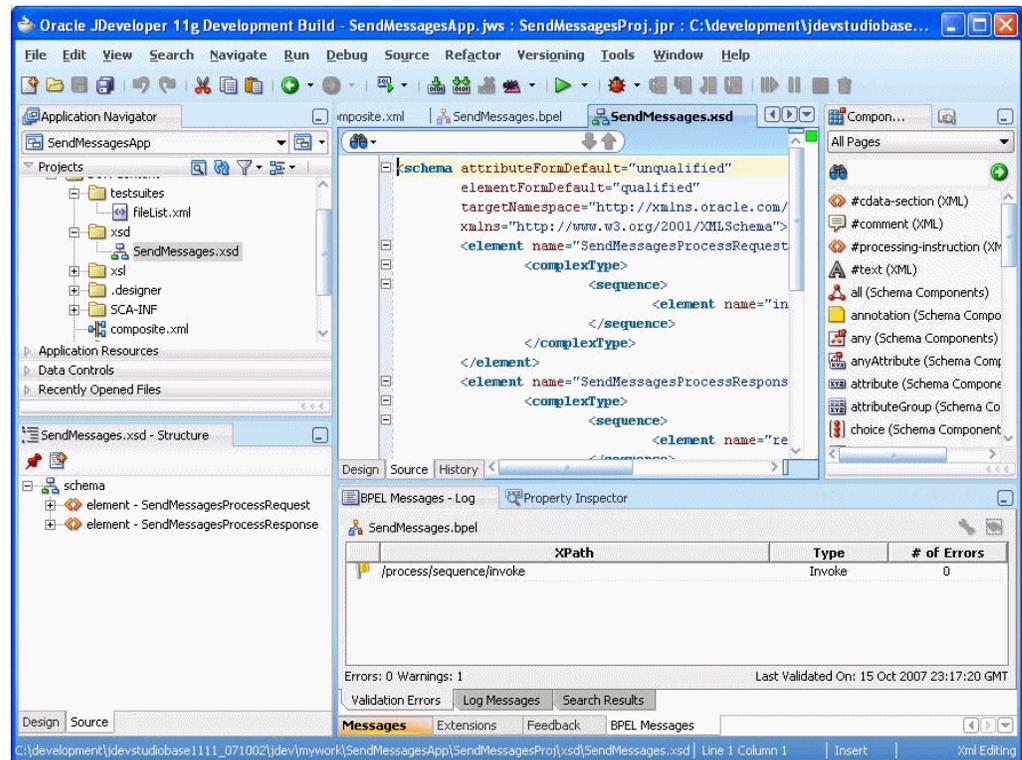
Note: You will later create the messaging service resource that is used to send the message when you create the User Notification BPEL process (steps 13-19).

Figure 41–7 Empty and Default BPEL Application



8. Expand the *xsd* folder in the Application Navigator and open *SendMessages.xsd* by double-clicking it .
9. Click on the **Source** tab (Figure 41–8).

Figure 41–8 Accessing the SendMessages.xsd File



10. Perform the following modifications to the inputs of this BPEL application:

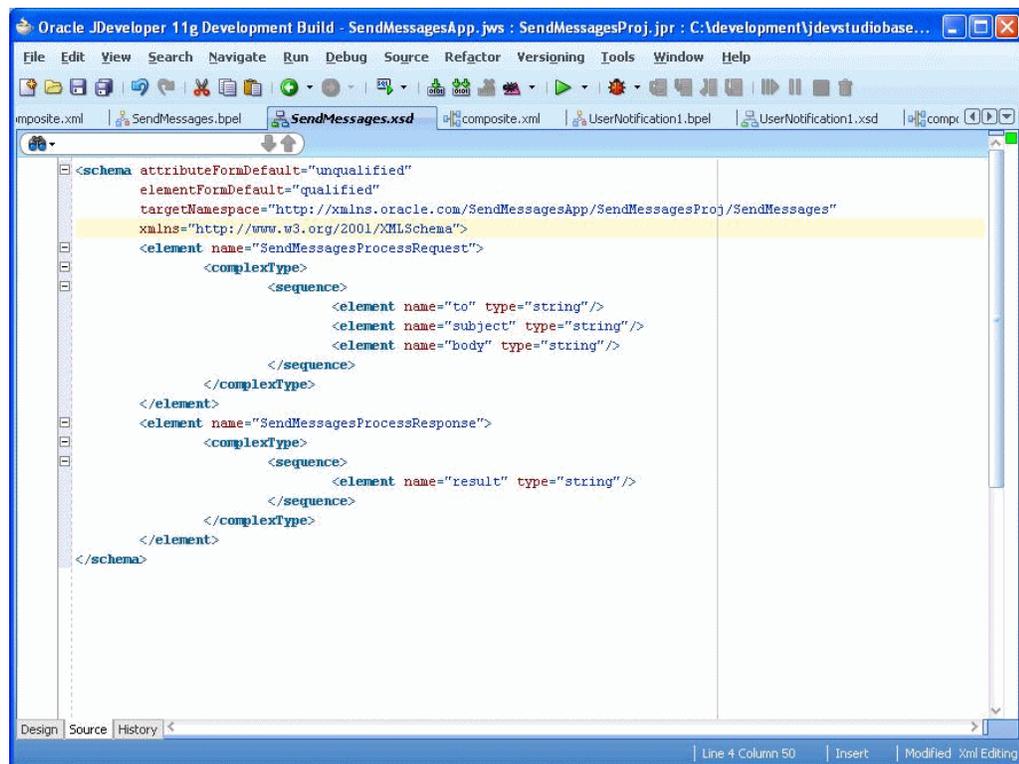
In the generated file, `SendMessage.xsd`, in the `xsd` folder in the application navigator under projects, the following element definition is created by default:

```
<element name="input" type="string"/>
```

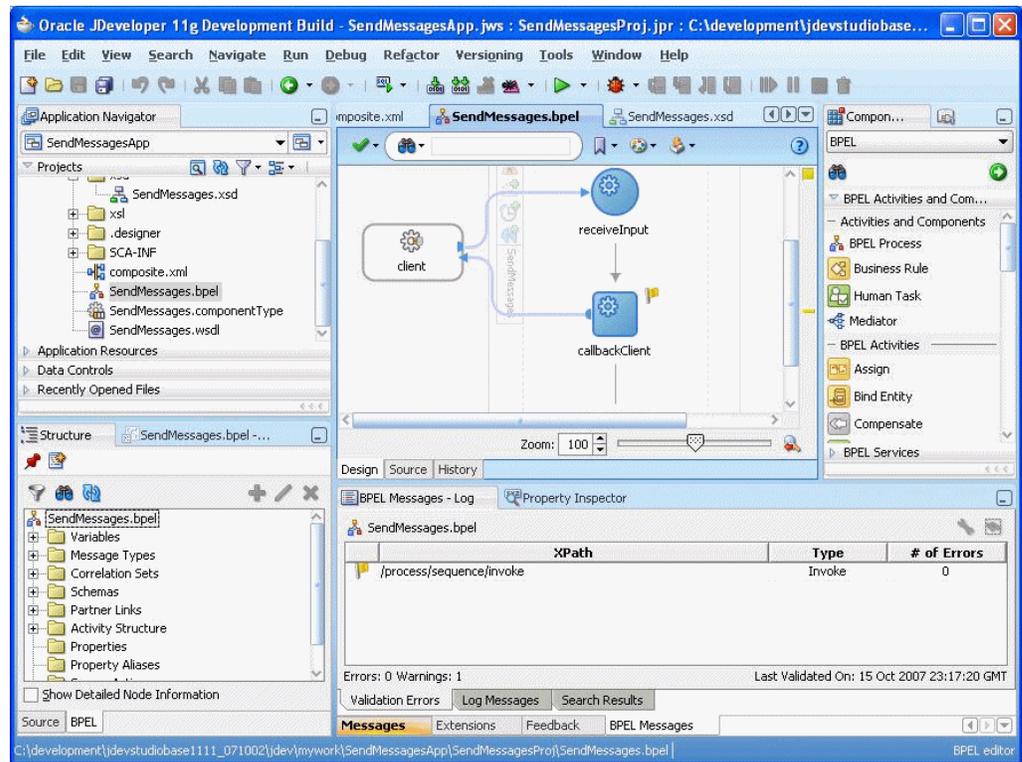
This xsd element defines the input for the BPEL process.

Select the *Source* tab (Figure 41–9), 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" />
```

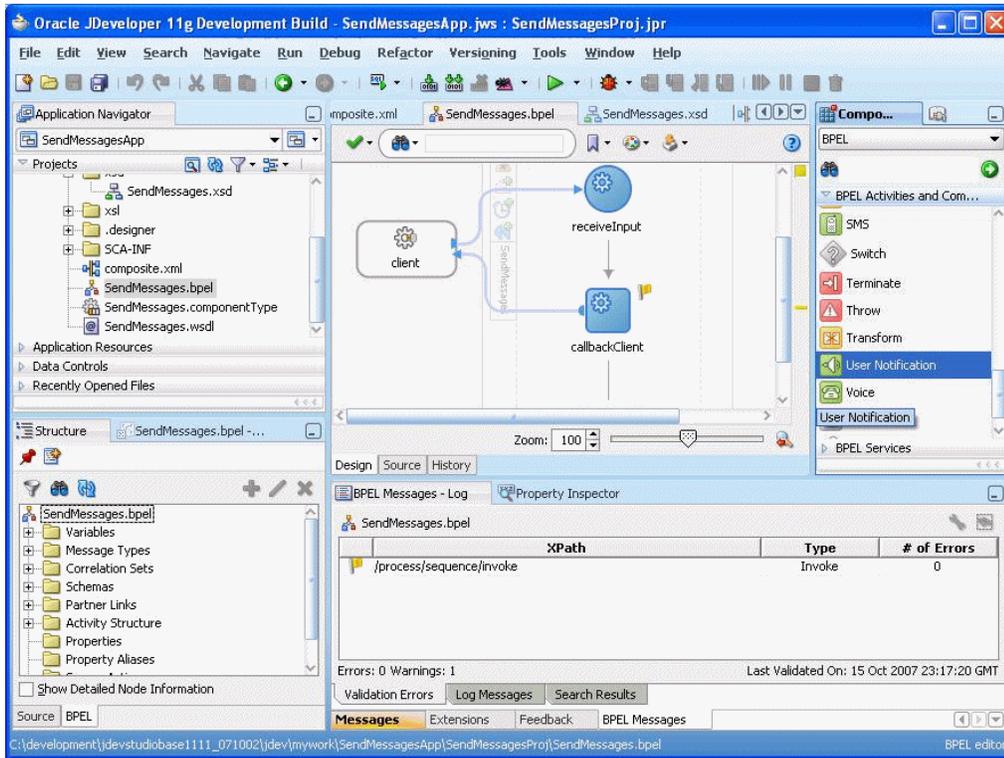
Figure 41–9 Modifying the Inputs in the *SendMessages.xsd* File

11. Perform a **Save All**.
12. Select the *SendMessage.bpel* editor screen from the Application Navigator (Figure 41–10).

Figure 41–10 Selecting *SendMessage.bpel* in the Application Navigator

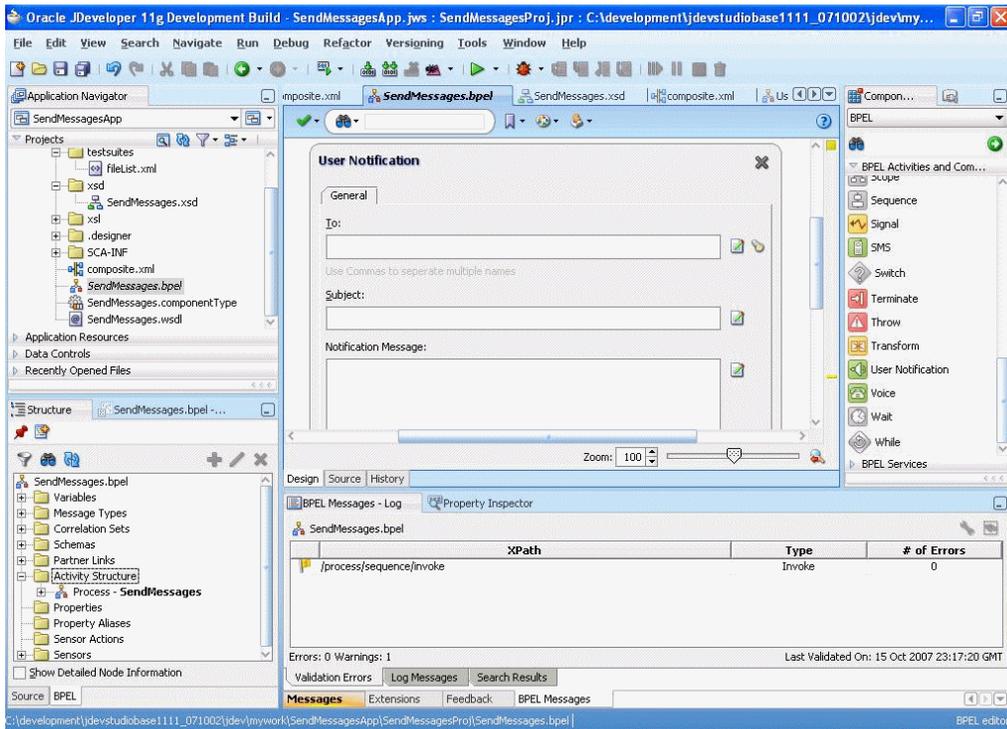
13. Drag and drop **User Notification** from *BPEL Activities and Components* located in the *Component Palette* between the *receiveInput* and *callbackClient* activities (Figure 41–11).

Figure 41–11 Drag and Drop the User Notification Activity from the Component Palette



The User Notification activity appears (Figure 41–12).

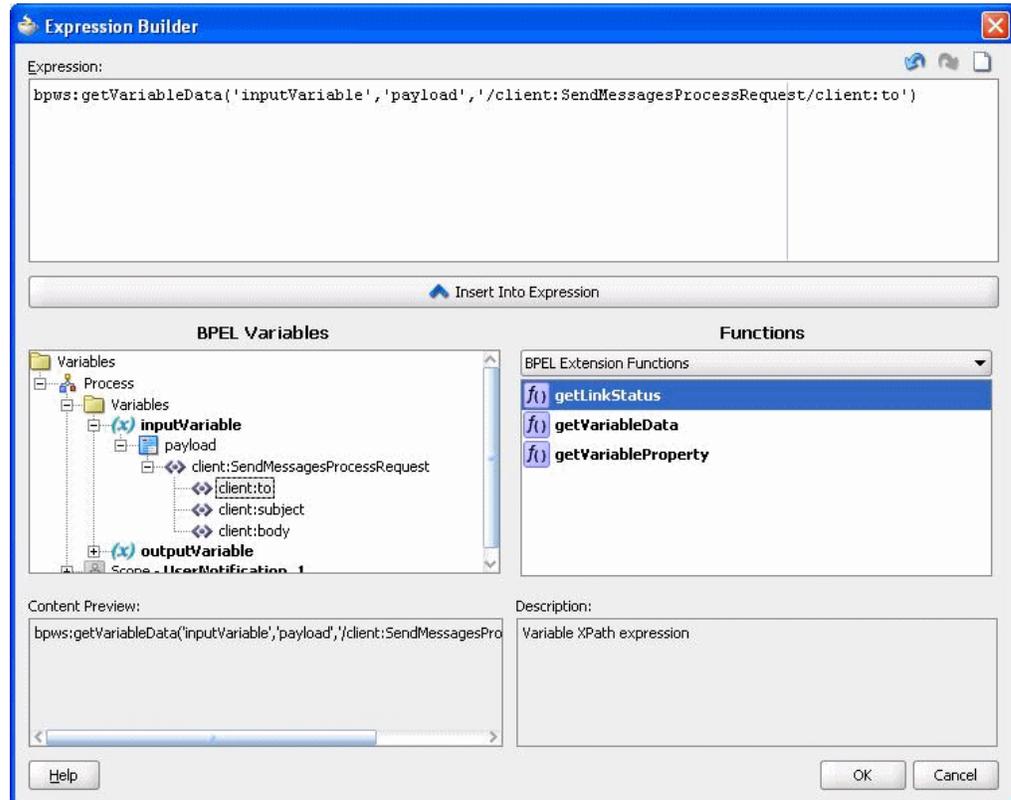
Figure 41–12 User Notification Activity Before Configuration of Inputs



14. Click the XPath Expression Builder icon to the right of the "To:" input box.

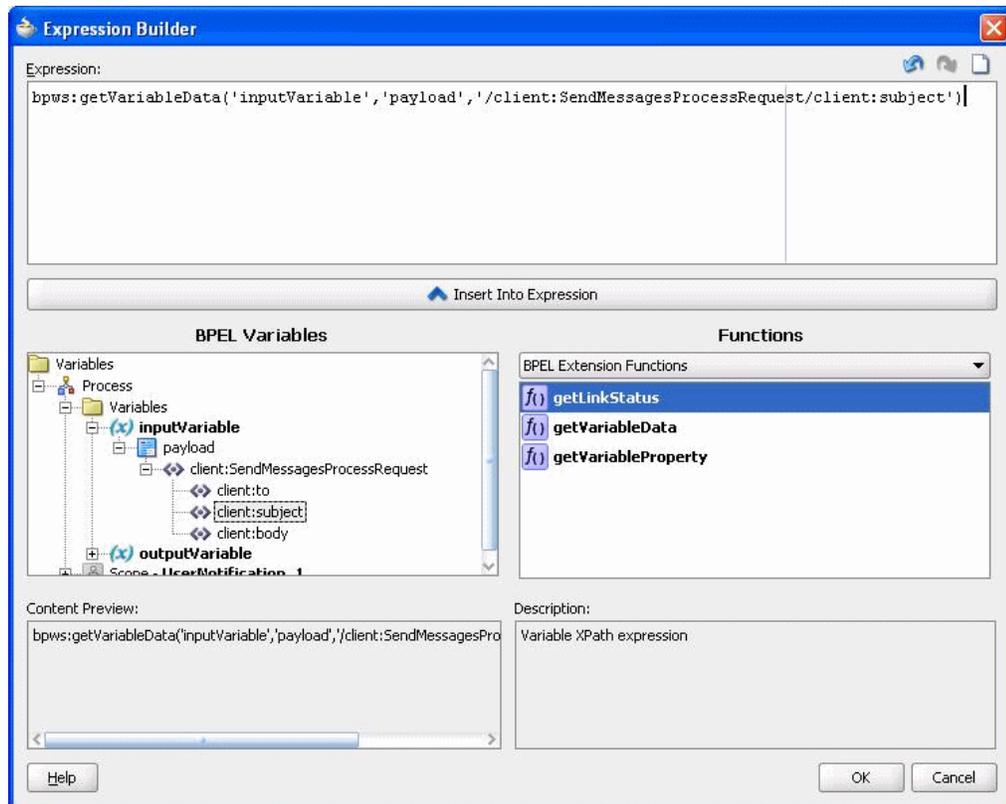
15. Modify the expression for the recipient, "to", as follows:
 - In the BPEL Variables pane, select **Variables, InputVariable, Payload, client:SendMessageProcessRequest**, and **client:to** (Figure 41–13).
 - Click **Insert Into Expression**.
 - Click **OK**.

Figure 41–13 Defining the Recipient ("to") Expression



16. Click the XPath Expression Builder icon to the right of the "subject:" input box.
17. Modify the expression for the *subject* as follows:
 - In the BPEL Variables pane, select **Variables, InputVariable, Payload, client:SendMessageProcessRequest**, and **client:subject** (Figure 41–14).
 - Click **Insert Into Expression**.
 - Click **OK**.

Figure 41–14 Defining the Subject Expression

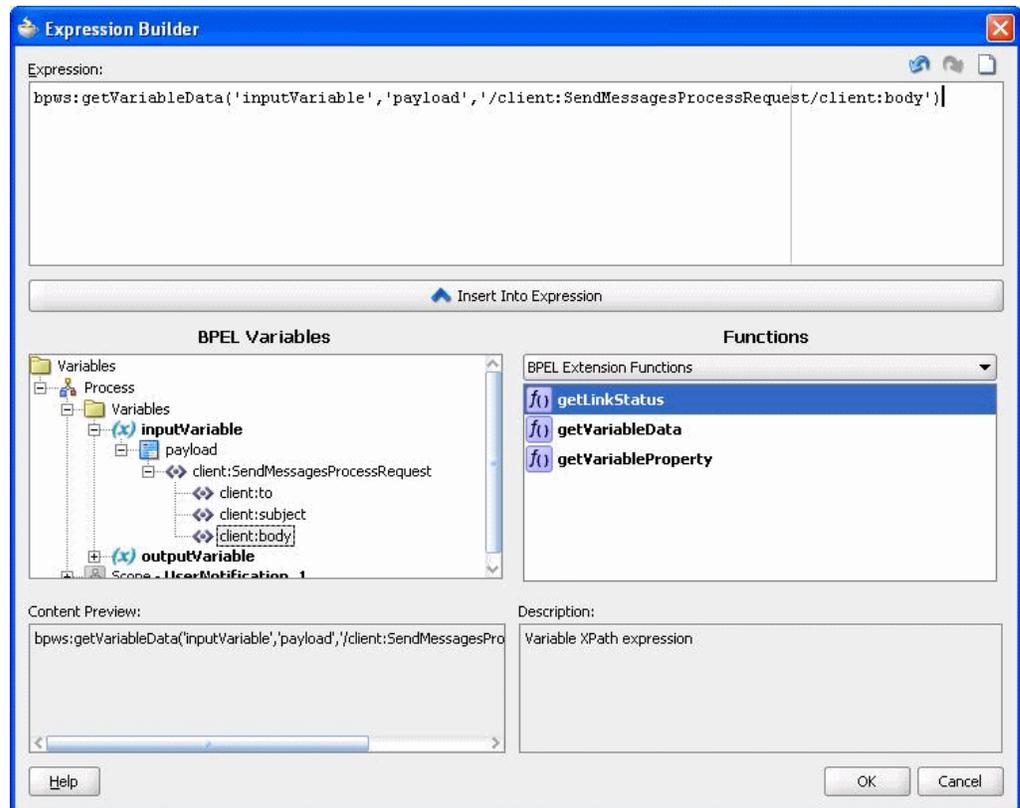


18. Click the XPath Expression Builder icon to the right of the "body:" input box.

19. Modify the expression for the *body* as follows:

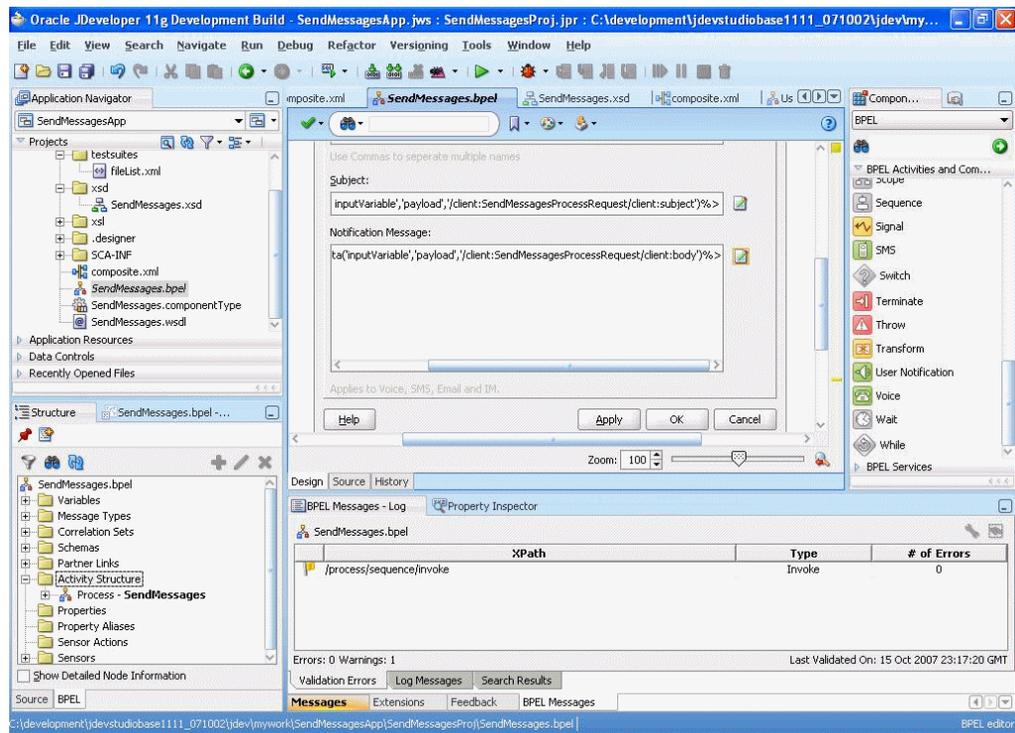
- In the BPEL Variables pane, select **Variables**, **InputVariable**, **Payload**, **client:SendMessageProcessRequest**, and **client:body** (Figure 41–15).
- Click **Insert Into Expression**.

Figure 41–15 Defining the Body Expression



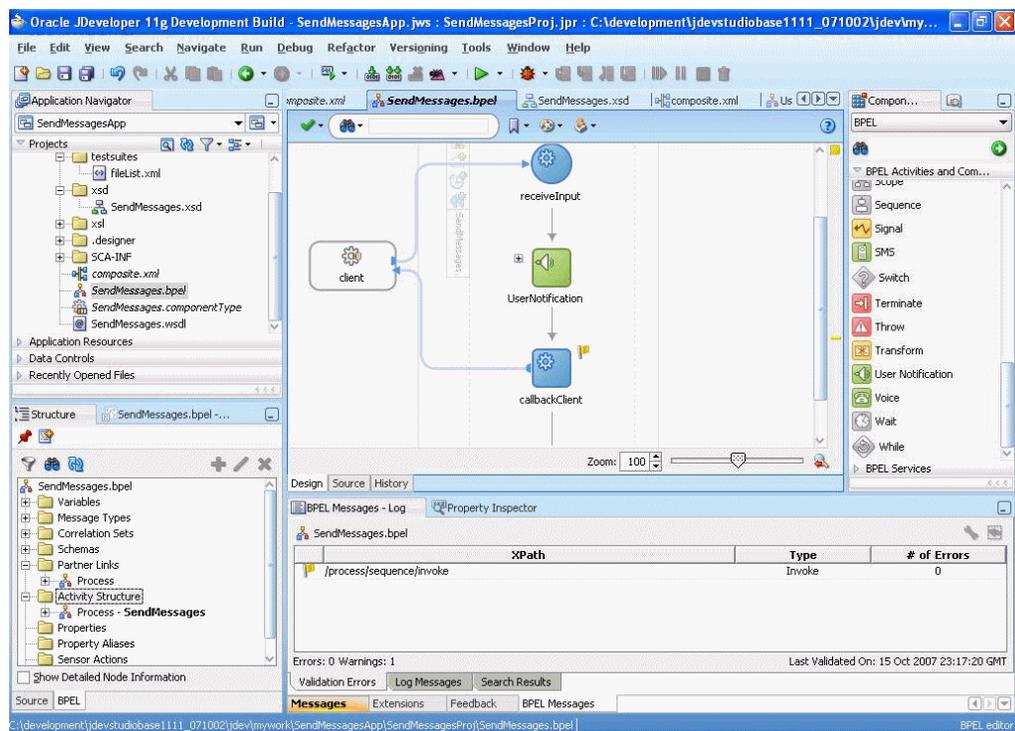
- Click **OK**.
- Click **OK** to apply the changes (Figure 41–16).

Figure 41–16 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 41–17). The SOA Composite is complete.

Figure 41–17 User Notification Activity After Configuration of Inputs

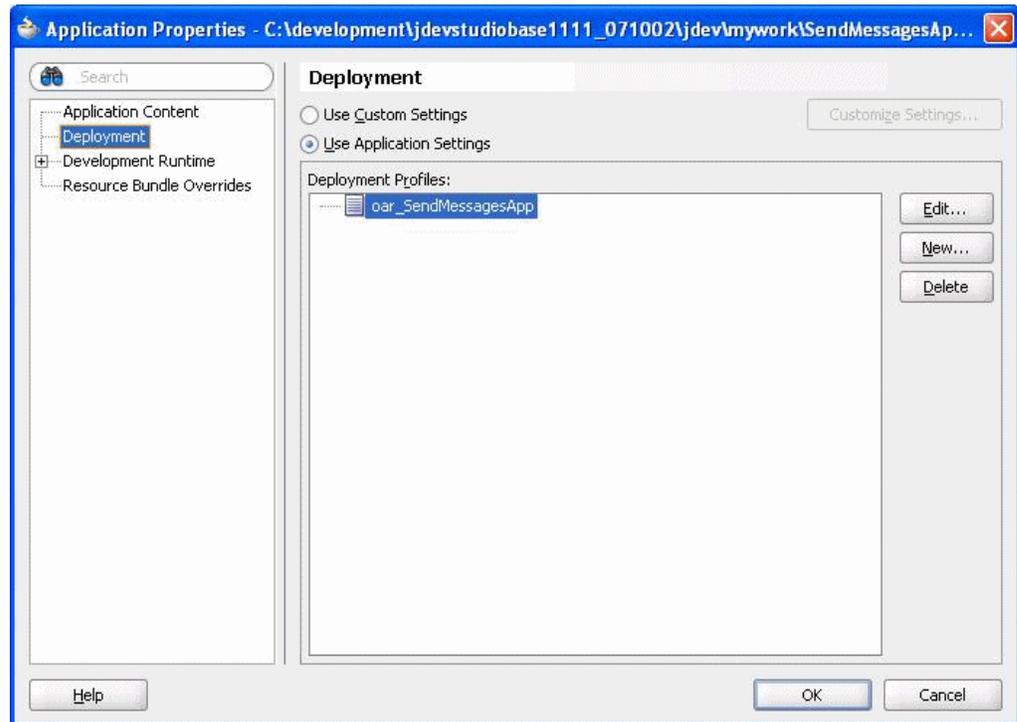


41.4 Creating a Deployment Profile

Perform the following steps to create a deployment profile:

1. Select the application, right-click, and select **Application Properties**.
2. Select **Deployment** and click **New** (Figure 41-18).

Figure 41-18 Creating a Deployment Profile (1 of 3)



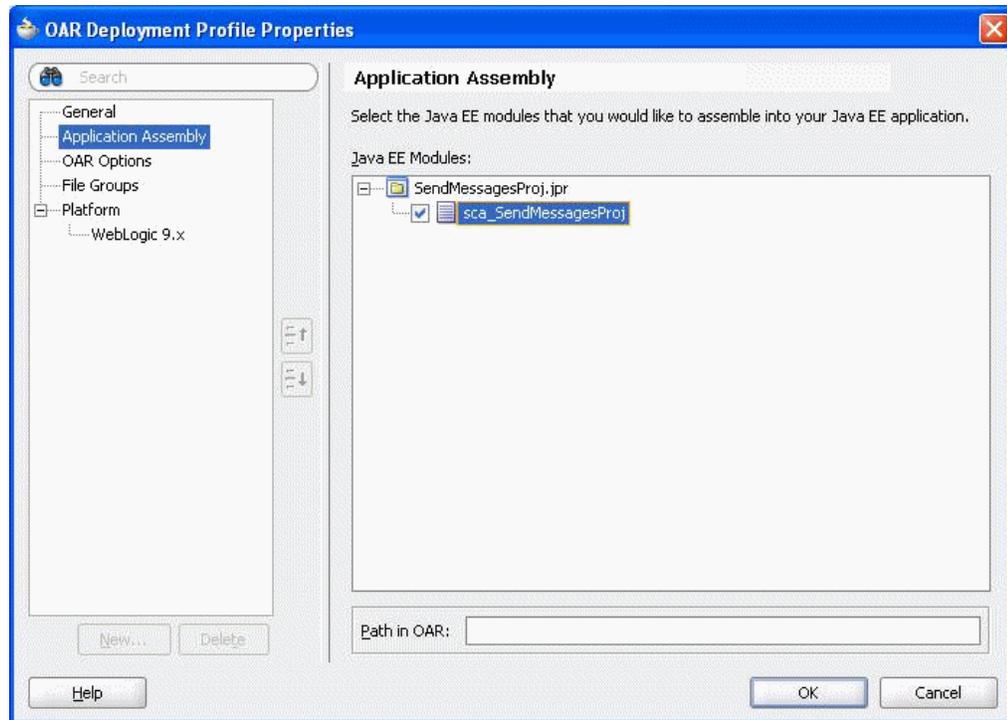
3. Select *OAR File* (Oracle Application archive) as the *Archive Type* (Figure 41-19).

Figure 41-19 Creating a Deployment Profile (2 of 3)



4. In the *Application Properties* window, click **Edit**.
5. Check *sca_SendMessagesProj* to be assembled into your application (Figure 41–20).

Figure 41–20 *Creating a Deployment Profile (3 of 3)*



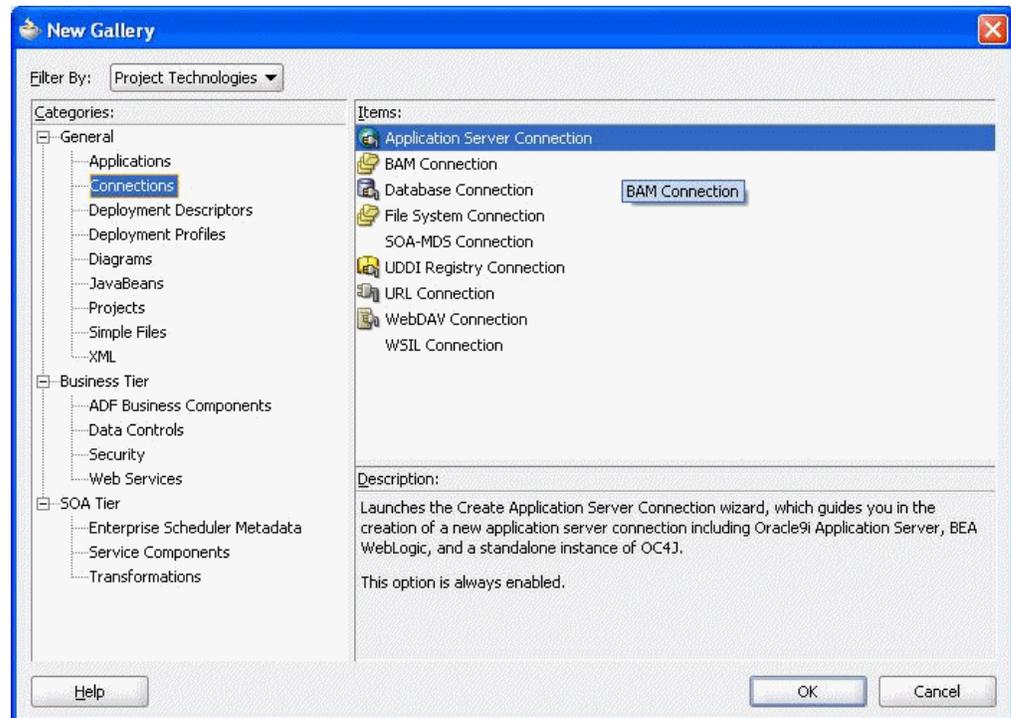
6. Click **OK** twice.
The deployment profile is complete.

41.5 Creating a New Application Server Connection

Perform the following steps to create a new Application Server Connection.

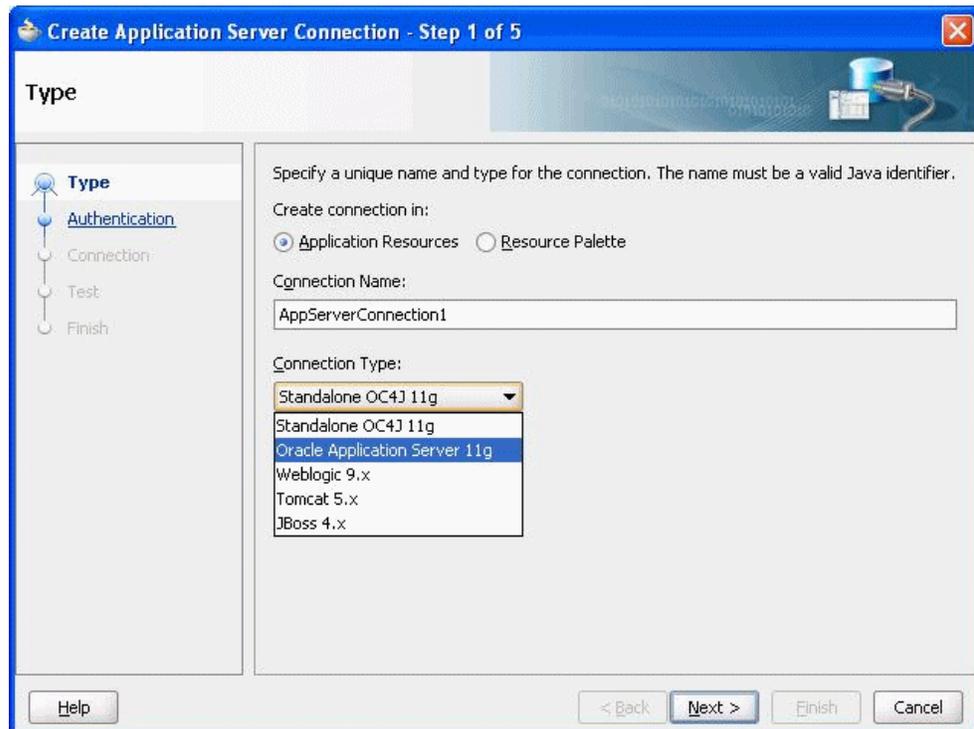
1. Create a new Application Server Connection by right-clicking the project and selecting **New, Connections, and Application Server Connection** (Figure 41–21).

Figure 41–21 New Application Server Connection



2. Name the connection “MyAppServerConnection” and click **Next** (Figure 41–22).
3. Select "Oracle Application Server 11g" as the *Connection Type*.

Figure 41–22 New Application Server Connection



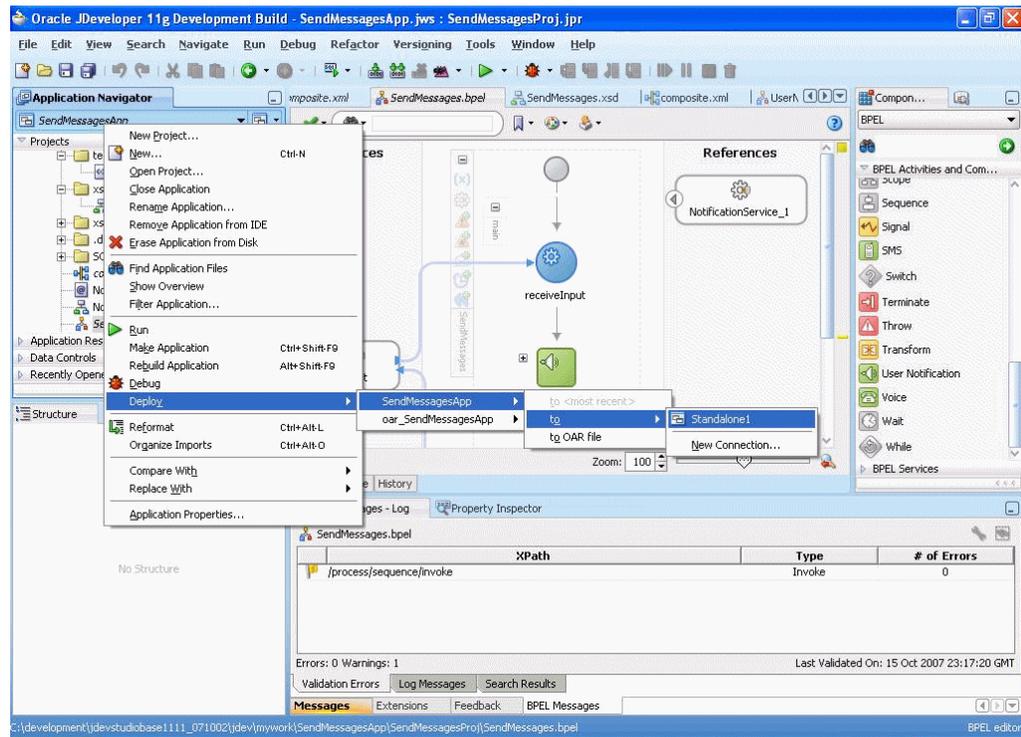
4. Enter the authentication information. The typical values are:
 Username: fmwadmin
 Password: welcome1
5. On the *Connection* screen, enter the hostname and click **Next**.
6. On the *Test* screen click **Test Connection**.
7. Verify that the message "Success!" appears.
 The Application Server Connection has been created.

41.6 Deploying the Application

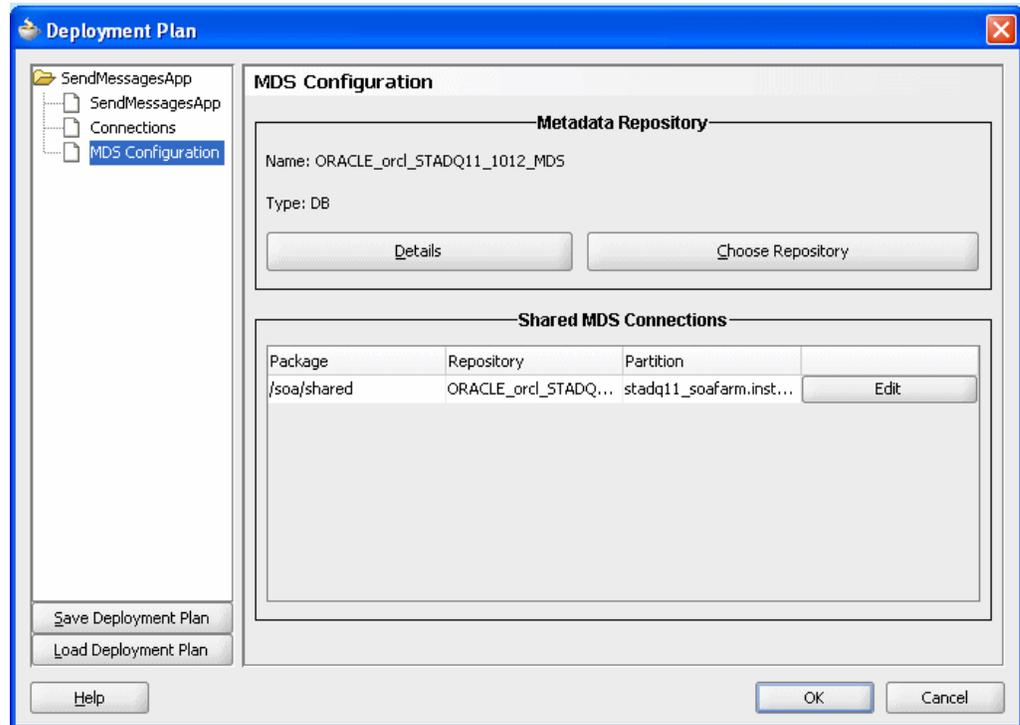
Perform the following steps to deploy the application:

1. Deploy the application by selecting **SendMessage Application, Deploy, SendMessageApp, to, and MyAppServerConnection** (Figure 41-23).

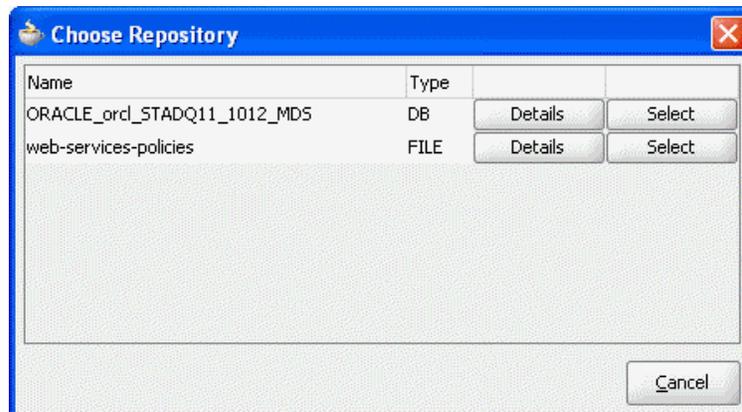
Figure 41-23 New Application Server Connection



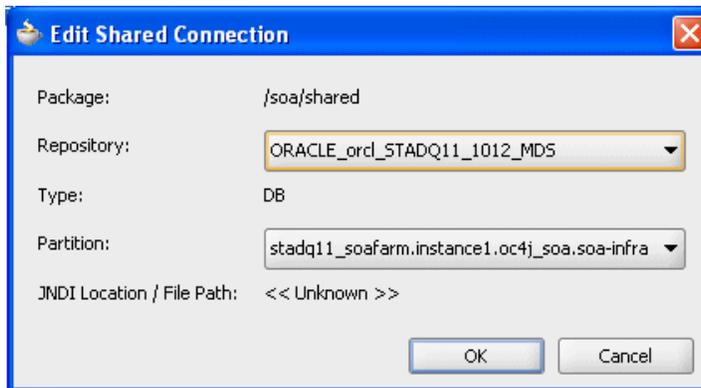
2. Verify that the message "Build Successful" appears in the log.
3. Enter the default revision and click **OK**.
4. In the *Deployment Plan* window select **MDS Configuration** (Figure 41-24).

Figure 41–24 Performing MDS Configuration

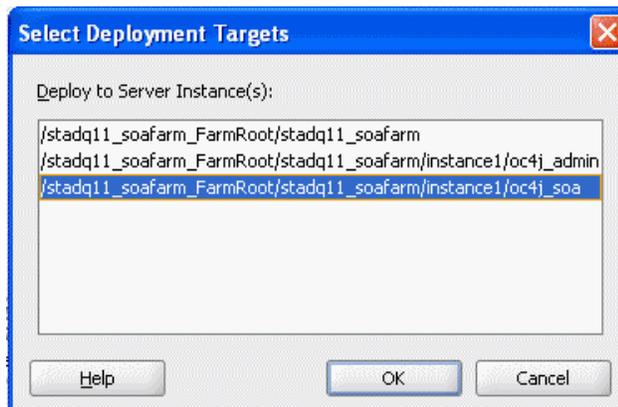
5. Click **Choose Repository** in the *Metadata Repository* box.
6. In the *Choose Repository* window, click **Select** next to the repository starting with "ORACLE" with *Type* "DB." For example, "ORACLE_orcl_STADQ11_1012_MDS" (Figure 41–25).

Figure 41–25 Choosing a Repository

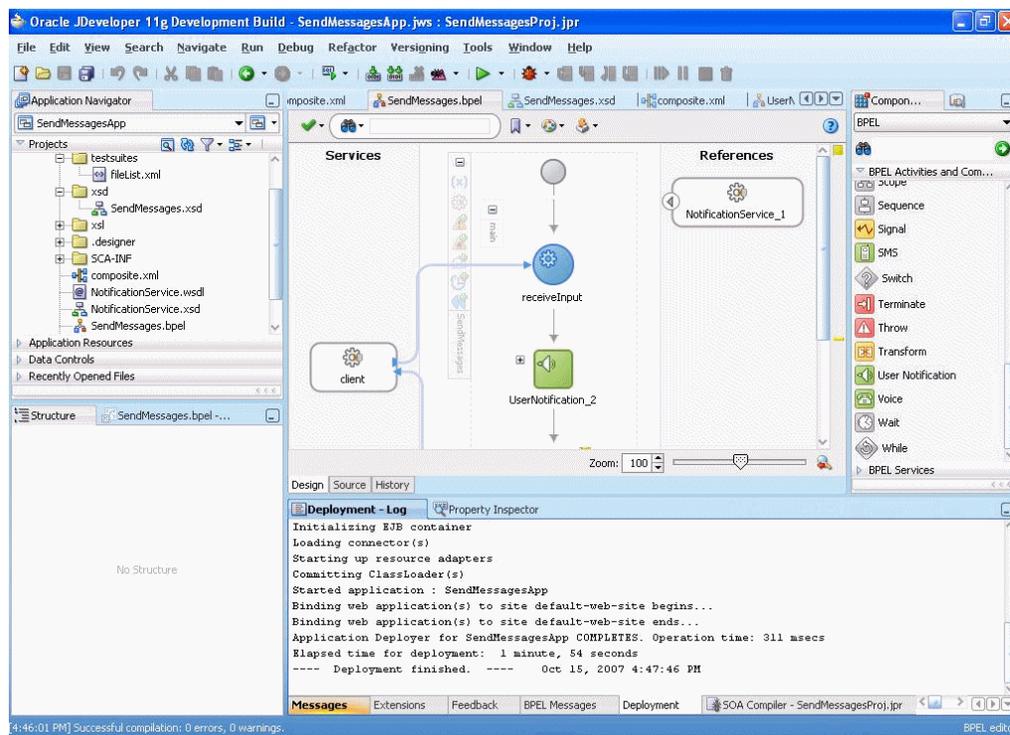
7. In the *Shared MDS Connections* box, select **Edit**.
8. In the *Partition* drop down menu, select the partition with the suffix *.soa-infra* only (Figure 41–27).

Figure 41–26 Editing a Shared Connection

9. Click **OK**.
10. Select the deployment target ending with "oc4j_soa" (Figure 41–27) and click **OK**.

Figure 41–27 Selecting a Deployment Target

11. Verify that the message "Deployment Finished" appears in the deployment log (Figure 41–28).

Figure 41–28 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 "Configuring Notifications" in the *Oracle Fusion Middleware SOA Developer's Guide* for more information.

41.7 Configuring Drivers

You have created and deployed the application, and configured the User Preferences. You must also configure drivers. For information on deploying drivers, refer to the

Perform the following steps to configure the drivers:

1. Log into the Enterprise Manager Fusion Middleware Control console as an administrator.
2. Expand the *Fusion Middleware* folder and then the *SOA* folder.
3. Navigate to the User Messaging Service *Home* page (`/instance/oc4j_soa/sdpmessagingserver`).
4. If needed, expand the *Associated Drivers* section of the User Messaging Service *Home* page.
5. Select the *Local* tab to access the drivers deployed to the SOA instance.
6. Select a driver and then click the adjacent **Edit** icon (a pencil) in the *Configure Driver* column.

The configuration page appears.

7. Expand the *Driver-Specific Configuration* section and configure the driver parameters as described in steps 8-9.
8. Configure the following parameters for e-mail:
 - IncomingMailServer -- The hostname of the incoming IMAP or POP3 mail server.
 - IncomingMailServerPort -- The port number of the IMAP or POP3 mail server.
9. Configure the following parameters for IM:
 - IMServerPassword -- A comma-separated list of passwords for each user name listed for the IMServerUsername property.
 - SmsAccountId -- The Account Identifier on the SMS-C (Short Message Service Center).
 - SmsAccountId -- The Account Identifier on the SMS-C (Short Message Service Center).
10. Configure the following parameters for the SMPP Driver (SMS):
 - IMServerHost -- The host name of the Jabber server. For multiple servers, use a comma-separated list (for example, enter my1.host.com, my2.host.com). If only one host name is specified, it will be used for all accounts.
 - IMServerPort -- The corresponding comma-separated list of Jabber server ports (for example, enter 5222, 5222)
 - IMServerUsername -- A list of Jabber user names for login. These user accounts will be automatically created, if necessary, on the corresponding Jabber servers. If you have multiple servers listed in the IMServerHost and IMServer Port properties, then there must be an equal number of user names (one user name per server). If you have only one server listed in these properties, then all of the user names listed in this property will use that server. For example, enter oracleagent1, oracleagent2. You can also enter a complete Jabber ID if its domain name is different from the Jabber server host name (for example, enter oracleagent1@host.com).
 - IMServerPassword -- A comma-separated list of passwords for each user name listed for the IMServerUsername property.

For more information on configuring driver parameters and setting up the environment, refer to "Configuring Notifications" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

41.8 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 into your SOA farm at the following URL:
http://<server>:<port>/sdpMessaging/userprefs-ui
The User Messaging Preferences application appears.
2. Click on the *Messaging Devices* tab (Figure 41–30).

Figure 41–29 Messaging Devices Tab

Channel	Address	Action
VOICE	3105555555	✗
SMS	3105555555	✗
IM	scot.tiger@oracle.com	✗

You are prompted for login credentials.

3. Enter the following login credentials:

Username: fmwadmin

Password: welcome1

4. In the Messaging Devices tab, select a device from the Device and Channel column, or create a new device by selecting **Create**.
5. Define the channels for the device by first selecting the transport type from the *Add Channels* list and then by entering the appropriate number or address.
6. Set a device as the default by expanding the device folder, and then clicking **Set as Default** adjacent to the selected device channel.

A checkmark appears next to the selected device channel, designating it as the default means of receiving notifications. All messages sent to that user will be sent to that channel.

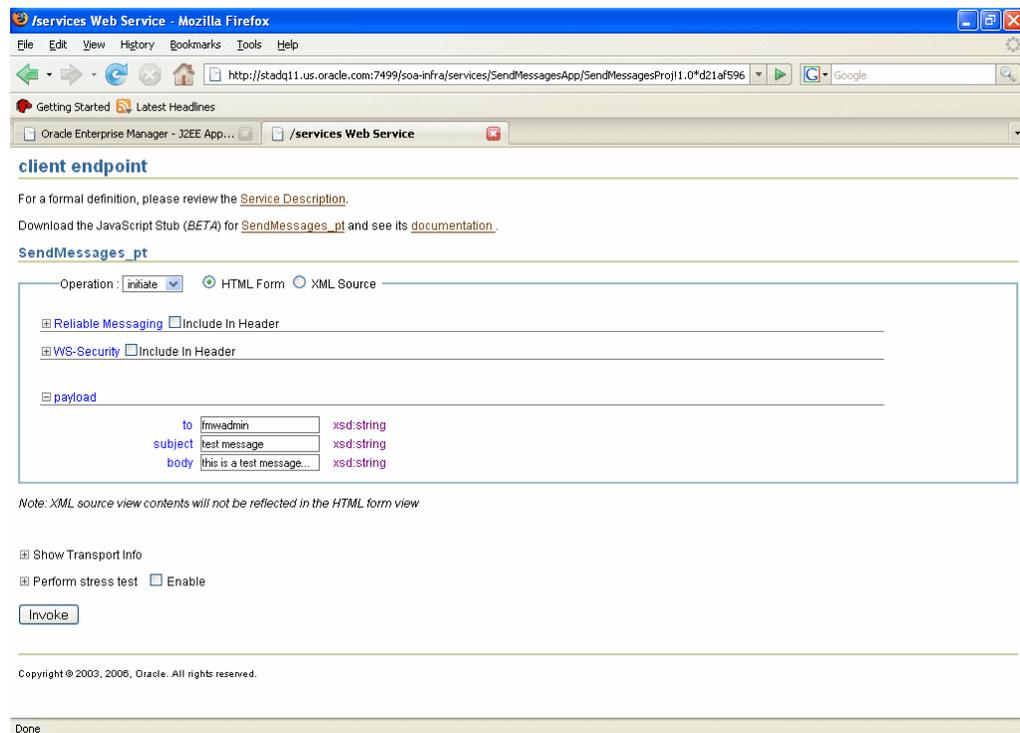
41.9 Testing the Sample

The following steps describe how to perform a test message transmission to through the *Client Endpoint* window.

Perform the following steps to run and test the sample:

1. Open a Web browser window and login to `http://<host>:<port>/soa-infra`
2. Select the SendMessagesApp **Test Client** link.

The *Client Endpoint* window appears (Figure 41–30).

Figure 41–30 Testing Messaging Using in the Client Endpoint Window

3. In the *Client Endpoint* window provide the input values for invoking *SendMessageApp*.

Enter the following values:

- `to:` `fmwadmin` (the user)
- `subject:` `notification test` (the subject)
- `body:` the message content

4. Click **Invoke**.

41.9.1 Verifying the Execution of Sending the E-mail

Log in to the device you configured as the preferred channel for the user in User Preferences. Verify that a message has been received by the device.

Oracle User Messaging Service Expense Report Use Case

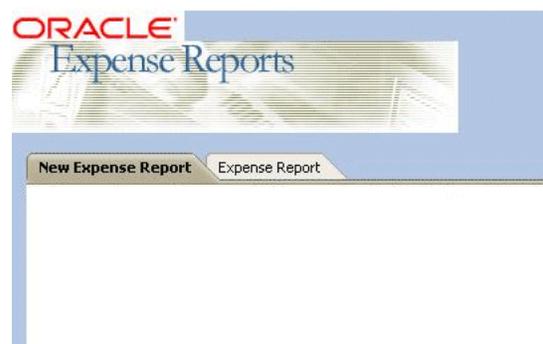
This document describes how to build an expense report use case through the following sections:

- Section 42.1, "Overview of the Expense Report Use Case"
- Section 42.2, "Deploying the Pre-Built Sample Applications"
- Section 42.3, "Running the Sample Application"
- Section 42.4, "Building the Expense Report Applications"

42.1 Overview of the Expense Report Use Case

This use case illustrates how to create an expense report that users can escalate through the management chain for ultimate approval. In this scenario, a user known as *jstein*, creates an expense report through Oracle Expense Reports (Figure 42–1) that is addressed to his manager, *cdickens*. Upon completing the expense report, the system creates an assigned task related to the expense report for *cdickens* in the Oracle BPM Worklist and also sends him an e-mail notification. Once he receives the e-mail, *cdickens* approves the expense report. His approval triggers the escalation of the expense report task to the next management tier, that of his director and fellow Worklist user, *wfaulkner*, who likewise receives an e-mail notification and a Worklist task that awaits his approval. Once *wfaulkner* approves the expense report task from the e-mail notification, it is resolved in the Workflow.

Figure 42–1 The Expense Report



The expense report use case is comprised of three applications, *ExpenseReportADFBCApp*, which defines the notification event using ADF BC (Application Development Framework Business Components),

ExpenseReportComposite, which defines a mediator that consumes the events from the ADF BC component and transforms them into BPEL input payload that invokes the human workflow, defined in *ExpenseAppHumanTaskFlow*.

42.1.1 Prerequisites

The expense report scenario requires the following:

- An IMAP4 or POP3 e-mail server.
- E-mail accounts provisioned to the e-mail server for users *jstein*, *cdickens*, and *wfaulkner*.
- The E-Mail Driver configured to point to the e-mail server. See [Section 42.1.1.2](#).
- Database tables that define the business components. See [Section 42.1.1.3](#).

42.1.1.1 Required Files

The following files are required:

- *expense_oracle.sql* ([Figure 42-1](#)) -- Defines the *expense_report* and *expense_items* schemas. Both *ExpenseReportADFBCApp* and *ExpenseAppHumanTaskFlow* require a SQL connection to these schemas. For information on creating this connection, refer to [Section 42.1.1.3](#).

Example 42-1 *expense_oracle.sql*

```
CREATE TABLE expense_report
  (id INTEGER PRIMARY KEY,
   creator VARCHAR2(100),
   created_date TIMESTAMP,
   cost_center VARCHAR2(100),
   reimb_currency VARCHAR2(100),
   purpose VARCHAR2(2000),
   status VARCHAR2(100)
  );
```

```
CREATE TABLE expense_items
  (expense_id INTEGER REFERENCES expense_report(id),
   receipt_date TIMESTAMP,
   receipt_amount NUMBER,
   expense_type VARCHAR2(100),
   justification VARCHAR2(2000)
  );
```

- *ExpenseReportADFBCApp* -- This includes *ExpenseReportADFBCApp*, a JDeveloper application containing the ADF BC component that defines the notification event.
- *ExpenseReportCompositeApp* -- This includes *ExpenseReportComposite*, a JDeveloper SOA composite application (SCA) that defines mediator and BPEL process and the JDeveloper task flow application, *ExpenseAppHumanTaskFlow*.

Note: The SCA and task flow application must be deployed to the same OC4J container. For more information, see [Section 42.2](#), "[Deploying the Pre-Built Sample Applications](#)"

- *ExpenseReport.edl* file ([Example 42-2](#)), located in the *ExpenseReportComposite* directory. This .edl (event definition language) file describes the Expense

Report's business events. [Example 42–2](#) defines `ExpenseEvent` as the event that the mediator component of the `ExpenseReportComposite` application subscribes to and publishes to the `ExpenseAppHumanTaskFlow` application.

Example 42–2 `ExpenseReport.edl`

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<definitions xmlns="http://schemas.oracle.com/events/edl"
             targetNamespace="/expensereportmodel/events/edl/ExpenseReport">
  <schema-import namespace="/expensereportmodel/events/schema/ExpenseReport"
                 location="ExpenseReport.xsd"/>
  <event-definition name="ExpenseEvent">
    <content xmlns:ns0="/expensereportmodel/events/schema/ExpenseReport"
             element="ns0:ExpenseEventInfo"/>
  </event-definition>
</definitions>
```

- `ExpenseReport.edl` also names the event's schema definition file, `ExpenseReport.xsd` ([Example 42–3](#)). This `.xsd` (XML Schema Definition) document, which is located in the `ExpenseReportComposite` directory, describes the event's payload.

Example 42–3 `ExpenseReport.xsd`

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<xs:schema targetNamespace="/expensereportmodel/events/schema/ExpenseReport"
           xmlns="/expensereportmodel/events/schema/ExpenseReport" elementFormDefault="qualified"
           attributeFormDefault="unqualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ExpenseEventInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Id" type="DecimalValuePair" minOccurs="1"/>
        <xs:element name="Creator" type="StringValuePair" minOccurs="1"/>
        <xs:element name="CreatedDate" type="DateTimeValuePair" minOccurs="1"/>
        <xs:element name="CostCenter" type="StringValuePair" minOccurs="1"/>
        <xs:element name="ReimbCurrency" type="StringValuePair" minOccurs="1"/>
        <xs:element name="Purpose" type="StringValuePair" minOccurs="1"/>
        <xs:element name="Status" type="StringValuePair" minOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="ValuePair" abstract="true"/>
  <xs:complexType name="DateTimeValuePair">
    <xs:complexContent>
      <xs:extension base="ValuePair">
        <xs:sequence>
          <xs:element name="newValue" minOccurs="0">
            <xs:complexType>
              <xs:complexContent>
                <xs:extension base="xs:anyType">
                  <xs:attribute name="value" type="xs:dateTime"/>
                </xs:extension>
              </xs:complexContent>
            </xs:complexType>
          </xs:element>
          <xs:element name="oldValue" minOccurs="0">
            <xs:complexType>
              <xs:complexContent>
                <xs:extension base="xs:anyType">
                  <xs:attribute name="value" type="xs:dateTime"/>
                </xs:extension>
              </xs:complexContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

```

        </xs:complexContent>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="StringValuePair">
    <xs:complexContent>
        <xs:extension base="ValuePair">
            <xs:sequence>
                <xs:element name="newValue" minOccurs="0">
                    <xs:complexType>
                        <xs:complexContent>
                            <xs:extension base="xs:anyType">
                                <xs:attribute name="value" type="xs:string"/>
                            </xs:extension>
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="oldValue" minOccurs="0">
                    <xs:complexType>
                        <xs:complexContent>
                            <xs:extension base="xs:anyType">
                                <xs:attribute name="value" type="xs:string"/>
                            </xs:extension>
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="DecimalValuePair">
    <xs:complexContent>
        <xs:extension base="ValuePair">
            <xs:sequence>
                <xs:element name="newValue" minOccurs="0">
                    <xs:complexType>
                        <xs:complexContent>
                            <xs:extension base="xs:anyType">
                                <xs:attribute name="value" type="xs:decimal"/>
                            </xs:extension>
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="oldValue" minOccurs="0">
                    <xs:complexType>
                        <xs:complexContent>
                            <xs:extension base="xs:anyType">
                                <xs:attribute name="value" type="xs:decimal"/>
                            </xs:extension>
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
</xs:schema>

```

42.1.1.2 Configuring the User Messaging Service E-Mail Driver to Send and Receive Messages

To enable users to receive and forward expense report notifications, use Oracle Enterprise Manager Fusion Middleware Control to configure the parameters of the User Messaging Service E-Mail driver to point to the appropriate e-mail server.

To configure the E-Mail Driver:

1. Log into the Enterprise Manager Fusion Middleware Control console as an administrator (that is, use *fnwadmin* as the user name and *welcome1* as the password).
2. Navigate to the User Messaging Service *Home* page.
3. If needed, expand the *Associated Drivers* section (Figure 42–2).

Figure 42–2 Associated Drivers

Name	Driver Type	Status	Configure Driver
/farm1_FarmRoot/farm1/asinst_1/oc4j_soa/default/sdpMessagingDriver-smpp	User Messaging SMPP Driver	↑	
/farm1_FarmRoot/farm1/asinst_1/oc4j_soa/default/sdpMessagingDriver-email	User Messaging Email Driver	↑	
/farm1_FarmRoot/farm1/asinst_1/oc4j_soa/default/sdpMessagingDriver-xmpp	User Messaging XMPP Driver	↑	

4. Select the E-Mail driver and then click the adjacent **Edit** icon (illustrated in Figure 42–3 as a pencil) in the *Configure Driver* column.

Figure 42–3 The Edit Icon



The configuration page for the E-Mail driver displays (Figure 42–4).

Figure 42–4 The Basic Configuration Page

Basic Configuration

Common Configuration

Supported Delivery Types: EMAIL

* Capability: BOTH

* Cost: 1

* Speed: 1

* Sender Addresses: askdemo@usunnbc29.us.oracle.cor

* Default Sender Address: askdemo@usunnbc29.us.oracle.cor

* Supported Protocols: *

* Supported Carriers: *

Supported Content Types: text/plain, text/html, multipart/mixed, multipart/all, multipart/related

Supported Status Types: DELIVERY_TO_GATEWAY_SUCCESS, DELIVERY_TO_GATEWAY_FAILURE, USER_REPLY_ACKNOWLEDGEMENT_SUCCESS, USER_REPLY_ACKNOWLEDGEMENT_FAILURE

Sending Queues Info: OraSDPM/QueueConnectionFactory:OraSDPM/Que

Driver-Specific Configuration

Name	Description	Mandatory	Value
MailAccessProtocol	E-mail receiving protocol. The possible values are IMAP and POP3. Required only if e-mail receiving is supported on the driver instance		IMAP
RetryLimit	This value specifies the number of times to retry connecting to the incoming mail server, if the connection is lost due to some reason. The default value is -1 which means no limit to the number of tries.		-1

- If needed, expand the *Driver-Specific Configuration* section and configure the E-Mail driver parameters listed in [Table 42–1](#) to point the driver to the proper e-mail server.

Table 42–1 E-mail Driver Parameters

Parameter	Value
IncomingMailIDs	A comma-separated list of e-mail addresses that correspond to user names. Each e-mail address must occupy the same position in the list as their user name counterparts (listed in <i>IncomingUserIDs</i>).
IncomingMailServer	The host name of the incoming mail server.
IncomingMailServerPort	The port number of the IMAP or POP3 mail server. For example, enter a standard port number for the IMAP server, such as 143 or 993 (for S-IMAP, IMAP over SSL). Likewise, enter a standard port number for the IMAP server, such as 100 or 995 (S-POP, POP over SSL).
IncomingMailServerSSL	Enter true to enable the Messaging Enabler to communicate using SSL (Secure Sockets Layer). If this property is set to true, then you must set the <i>IncomingMailServerPort</i> to a port commonly used for SSL, such as 993.
IncomingUserIDs	A comma-separated list of user names of the mail accounts from which the E-Mail Driver instance polls.
IncomingUserPasswords	A comma-separated list of passwords that correspond to the user names. Each e-mail address must occupy the same position in the list as their user name counterparts (listed in <i>IncomingUserIDs</i>).
OutgoingDefaultFromAddr	The default <i>from</i> address of the e-mail (if none is provided in the message).
OutgoingMailServer	The name of the SMTP server.
OutgoingMailServerPort	The port number of the outgoing SMTP server.
OutgoingMailServerTLS	If set to <i>true</i> , then the Messaging Enabler uses TLS (Transport Layer Security) encryption when communicating with the SMTP server.
OutgoingPassword	The password used for SMTP authentication.
OutgoingUsername	The user name used for SMTP authentication.
ReceiveFolder	The name of the folder (such as <i>INBOX</i>) from which the driver polls messages.

42.1.1.3 Creating the Database Tables that Define the Business Components

Both *ExpenseReportADFCAApp* and *ExpenseAppHumanTaskFlow* require a connection to the application database that stores the `expense_report` and `expense_items` tables ([Example 42–1](#)). The connection to these tables enables *ExpenseReportADFCAApp* to provide the current and past expense report data to Oracle Expense Reports ([Figure 42–1](#)). *ExpenseAppHumanTaskFlow* also consumes the data from these tables to populate the payload information displayed in the view object.

Before you create the database tables, you must first create a database connection, which is a two-step process:

- Creating a new user ([Section 42.1.1.3.1](#)).

Connect to the database as `SYSDBA` and then create the "expense" user and password used to connect with the database containing the `expense_report` and `expense_items` schema.

2. Editing the existing database connection (Section 42.1.1.3.2).

Substitute the user name and password values in the existing SQL connection for those of the new user. You may also need to change the host and port values used for the existing SQL connection.

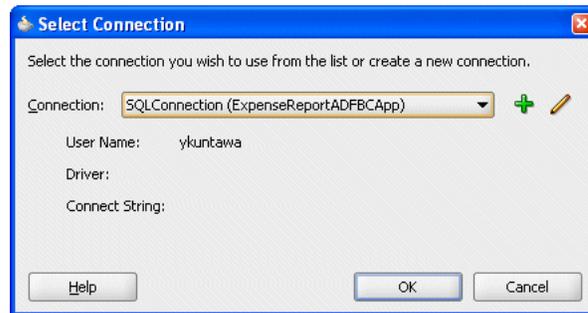
Note: To complete the Expense Report use case, you must edit the connections that ship with both *ExpenseAppHumanTaskFlow* and *ExpenseReportADFBCApp*.

After you create new user and password and edit the existing SQL connection, you create the Expense Report schema. For more information, see "Creating the Database Schema".

42.1.1.3.1 Creating a New User

1. In JDeveloper, open *ExpenseReportADFBCApp.jws*.
2. Select **Tools** and then **SQL Worksheet**. The *Select Connection* dialog appears (Figure 42-5).

Figure 42-5 The Select Connection Dialog



3. Click the **Add** icon (represented as a green plus sign in Figure 42-6). The *Create Database Connection* dialog appears.

Figure 42-6 The Add Icon



4. Complete the dialog by entering the following:
 - Select **Application Resources** (the default).
 - A connection name, such as *SysDBAConnection*.
 - Enter `SYSDBA` as the user name and enter the administrator password.
 - Enter the host name.
 - Enter *orcl* for the SID
5. Click **Test Connection**. If you defined the database connection properly, *Success!* appears.

6. Click OK.
7. Create a user by entering a script similar to [Example 42-4](#) in the *Enter SQL Statement* pane ([Figure 42-7](#)). [Example 42-4](#) sets *expense* as both the user name and password to the database.

Example 42-4 Creating a User

```

define bpm_user=expense
define bpm_passwd=expense
define def_tablespace=system
define temp_tablespace=temp

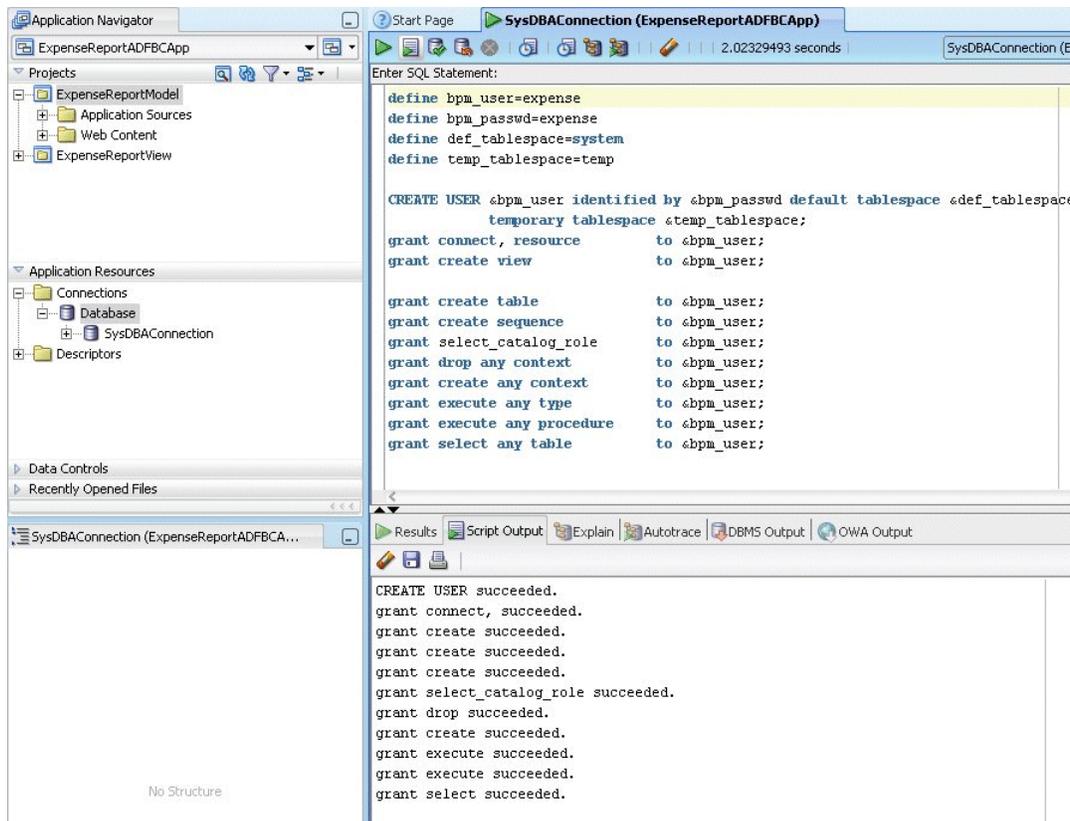
CREATE USER &bpm_user identified by &bpm_passwd default tablespace &def_tablespace
        temporary tablespace &temp_tablespace;

grant connect, resource      to &bpm_user;
grant create view           to &bpm_user;

grant create table          to &bpm_user;
grant create sequence       to &bpm_user;
grant select_catalog_role   to &bpm_user;
grant drop any context      to &bpm_user;
grant create any context    to &bpm_user;
grant execute any type      to &bpm_user;
grant execute any procedure to &bpm_user;
grant select any table      to &bpm_user;

```

Figure 42-7 Entering the Create User SQL Script



8. Click **Execute Script** (Figure 42–8). Data returned by the script displays in the *Script Output* pane (illustrated in Figure 42–7).

Figure 42–8 The Execute Script Icon



42.1.1.3.2 Editing the Existing Database Connection

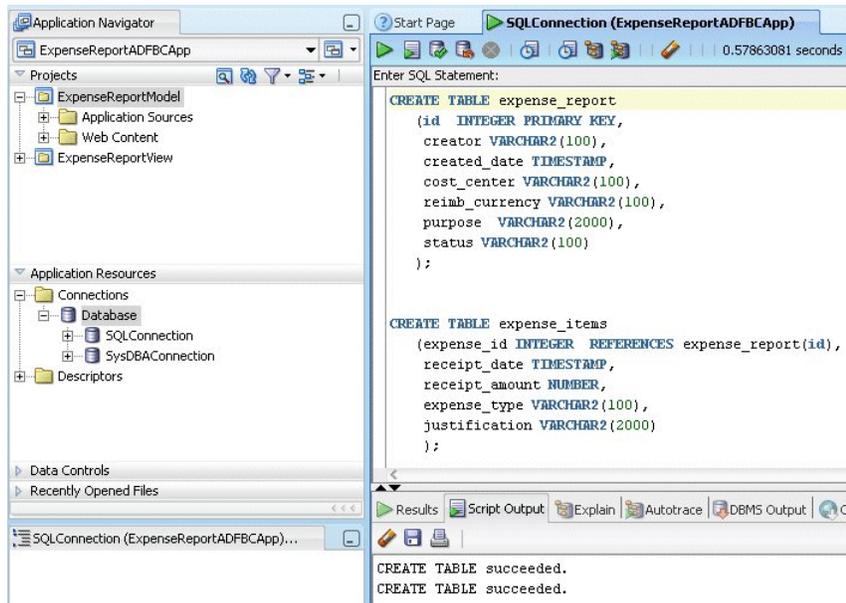
To create this connection and then seed the database tables:

1. In JDeveloper, click **Tools** and then **SQL Worksheet**. The *Select Connection* dialog appears (Figure 42–5).
2. Select the connection that ships with the ADF BC application, **SQLConnection(ExpenseReportADFBCApp)**, and then click **Edit** (Figure 42–3). The *Create Database Connection* dialog appears, populated with default values.
3. Replace the user name and password with those set for the expense user. For example, enter *expense* for the user name and password.
4. If applicable, enter a different host name.
5. Click **Test Connection**. If you defined the database connection properly, *Success!* appears.
6. Click **OK**.

Note: You must replace the default user names and passwords for the SQL connections that ship with the *ExpenseReportADFBCApp* application and the task flow application, *ExpenseAppHumanTaskFlow* with those set for the “expense” user. You can edit the connections through the *Edit Database Connection* dialog (accessed by right-clicking the *SQL Connections* node of *Application Resources*, such the one illustrated in Figure 42–9 and then by selecting **Properties** from the context menu).

Creating the Database Schema

1. As the new user, create the database tables by pasting the *expense_oracle.sql* (Example 42–1) into the *Create SQL Statement* pane.

Figure 42–9 Creating the Schema

2. Click **Execute Script** (Figure 42–8). Data returned by the script displays in the *Script Output* pane.

42.2 Deploying the Pre-Built Sample Applications

This section describes the following:

- [Deploying the SOA Composite and Task Flow Applications \(ExpenseReportCompositeApp\)](#)
- [Deploying the ADF BC Application \(ExpenseReportADFBCApp\)](#)

42.2.1 Deploying the SOA Composite and Task Flow Applications (ExpenseReportCompositeApp)

Deploy *ExpenseReportComposite* as follows:

1. Open the .jws file in JDeveloper. The JDeveloper main window appears.

Note: A dialog box may appear to warn you that the data from the project will be migrated to the latest Oracle JDeveloper version and that you will not be able to open the application or its project using an older release. Click **Yes**.

2. Create a new connection by first clicking **File** and then selecting **Connections** (located in the *General* section).
3. Select **Application Server Connection** from the *Items* pane and then click **OK**. The *Welcome* page of the Application Server Connection Wizard displays.
4. Complete the wizard as follows:
 - a. Click **Next**. The *Type* page appears.
 - b. Enter *11g* in the *Connection Name* field.

- c. Select **Oracle Application Server 11g** as the *Connection Type*. Click **Next**. The *Authentication* page appears.
 - d. Enter the user name and password (*fmwadmin* and *welcome1*, respectively).
 - e. Click **Next**. The *Connection* page appears.
 - f. Enter the server name and port.
 - g. Click **Next**. The *Test* page appears. Click **Test Connection**. If you defined the connection properly, *Success!* appears.
5. In the Application Navigator, right-click *ExpenseReportCompositeApp*.
 6. In the context menu that displays, select **Deploy > ExpenseReportSCApp > to** and then select the Oracle Application Server 11g connection.

Note: Verify that the *Build Successful* message displays in the log.

7. Accept the default value of 1.0 as the *New Version ID* in the *Revision ID* dialog and click **OK**.
8. In the *Deployment Plan* page select **MDS Configuration**.
9. Click **Choose Repository** in the *Metadata Repository* section.
10. In the *Choose Repository* window, click **Select** (next to the repository starting with "ORACLE" with *Type* "DB." For example, "ORACLE_orcl_STADQ11_1012_MDS".
11. In the *Shared MDS Connections* pane, select **Edit**.
12. In the *Partition* drop down menu, select the partition with the suffix *.soa-infra* only.
13. Click **OK**.
14. Select the deployment target ending with "oc4j_soa" and then click **OK**.
15. Verify that the *Deployment Finished* message displays in the deployment log.

42.2.2 Deploying the ADF BC Application (ExpenseReportADFBCApp)

To deploy *ExpenseReportADFBCApp*:

1. In the Application Navigator, right-click the *ExpenseReportModel* and then select **Deploy > ExpenseReportTaskFlowApp > to >** then select the Oracle Application Server 11g connection.
2. Verify that the *Deployment Finished* message displays in the deployment log.

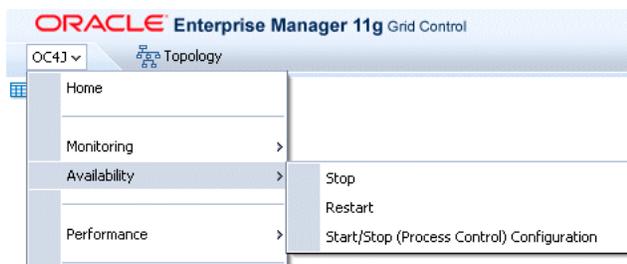
42.3 Running the Sample Application

Once you have deployed the Expense Report applications, you must restart the OC4J instance to which it has been deployed. To do this:

1. Log into the Enterprise Manager Fusion Middleware Grid Control console as an administrator (that is, use *fmwadmin* as the user name and *welcome1* as the password). Enterprise Manager Fusion Middleware Grid Control displays. The *Farm* tab also displays (Figure 42–10).

Figure 42–10 The Farm Tab

2. Expand *Fusion Middleware* and then expand *Application Servers*.
3. Expand *OC4Js* and then select the OC4J SOA instance that runs the Expense Report applications. The *OC4J* tab appears.
4. From the *OC4J* tab menu, select **Availability** (Figure 42–11).

Figure 42–11 The OC4J Menu

5. From the context menu, select **Stop** and then **Restart**.
6. Navigate to the following URL:

`http://hostname:port/fusion/sample/faces/expenseReport.jspx`

Note: Use either Internet Explorer 7.0 or the latest version of Mozilla Firefox (such as 5.0), to run this application.

7. Click the *Expense Report* tab (Figure 42–1)
8. Enter a unique ID, enter *jstein* as creator and enter values for *createdDate* and other fields.
9. Click **Next** and then add *ExpenseItems*.
10. Click **Next** and then **Commit**.
11. In a browser, open the Oracle BPM Worklist by entering following URL:

`http://hostname:port/integration/worklistapp`
12. Open the expense report approval e-mail sent to as *cdickens*.
13. Click the *Details* link.
14. Click **Approve**.
15. Open the expense report approval e-mail sent to *wfaulk* (user name).
16. Click the *Details* link.
17. Click **Approve**.
18. Check the status of the of the task in the Worklist application.

42.4 Building the Expense Report Applications

The following sections describe how to build the Expense Report applications:

- [Section 42.4.1, "Creating the ADF BC Application \(ExpenseReportADFBCApp\)"](#)

This section describes building the ADF BC (Application Development Framework Business Components) application which defines the business components and their corresponding events. The business components are populated from the `expense_report` and `expense_items` tables and views in the application database (Example 42-1). This application raises an event upon the commit or update actions of the ADF objects. This application is comprised of two projects, `ExpenseReportModel` and `ExpenseReportView`.

- [Section 42.4.2, "Creating the SOA Composite Application \(ExpenseReportComposite\)"](#)

This section describes building the SCA (SOA Composite Application) that subscribes to the events raised by `ExpenseReportADFBCApp` (the ADF BC layer). This application contains the mediator component, which consumes events, the BPEL process, and the human workflow, which assigns tasks to users or roles.

- [Section 42.4.3, "Creating the Task Flow Application \(ExpenseAppHumanTaskFlow\)"](#)

This section describes building the task display form, which includes adding the `approve`, `reject`, and `escalate` task operations.

42.4.1 Creating the ADF BC Application (ExpenseReportADFBCApp)

To create the application:

1. Click **File** and then click **New**. The *New Gallery* appears.
2. From the **Categories** pane, expand **General** and then select **Applications**.
3. In the *Items* pane, select **Application** and then click **OK**. The *Create Application Dialog* appears.

Note: **No Template [All Technologies]** (the default setting) must be selected from the Application Template list.

4. Enter `ExpenseReportADFBCApp` in the *Application Name* field. If needed, enter an alternate location for the application in the *Directory* field. Otherwise, accept the default location (`C:\JDeveloper\mywork\ExpenseReportADFBCApp`) and click **OK** to create the application. The *Create Project* dialog appears.
5. Click **Cancel** to dismiss the *Create Project* dialog. `ExpenseReportADFBCApp` appears in the Application Navigator.

42.4.1.1 Creating the Business Component Project

To create the business component project:

1. Click **File** and then **New**. The *New Gallery* appears.
2. Click **Projects** (located under the *General* node) and then click **Business Components Project** in the *Items* pane and then click **OK**. The *Welcome* page of the *Create Business Components Project* wizard appears.
3. Click **Next** to navigate to the *Location* page.

4. In the *Location* page, enter *ExpenseReportModel* in the *Project Name* field. If needed, enter an alternate location for the application in the *Directory* field. Otherwise, accept the default location
(*C:\JDeveloper\mywork\ExpenseReportADFBCApp\ExpenseReportModel*).
5. Click **Next**. The *Paths* page appears with values populated to the *Default Package*, *Java Source Path* and *Output Directory* fields.
6. If needed, change the information populated to the *Paths* page. Otherwise, click **Next**. The *Summary* page appears.
7. Click **Finish** to complete the wizard. *ExpenseReportModel* displays in the Application Navigator. Simultaneously, the *Initialize Business Components Project* dialog appears.

42.4.1.1.1 Creating the Database Connection To create the database connection:

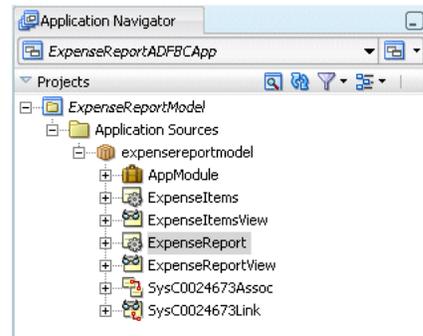
1. In the *Initialize Business Components Project* page, click the **Add** icon (represented as a green plus sign in [Figure 42-6](#)) next to the *Application Connection* list. The *Create Database Connection* page appears.
2. Complete the *Database Connection* page as follows:
 - Select **Application Resources** (the default).
 - Enter *SQL Connection* in the **Connection Name** field.
 - Select **Oracle (JDBC)** (the default) from the *Connection Type* list.
 - Enter the user name and password to the database. You must use the "expense" user name and password. For more information, see [Section 42.1.1.3.1](#).
 - Select **thin** as the driver type.
 - Enter the host name
 - Enter *orcl* as the SID.
3. Click **Test Connection**. If you defined the database connection properly, *Success!* appears.
4. Click **OK**. The *Create Business Components from Tables* wizard appears and opens to the *Entity Objects* page (Step 1).

42.4.1.1.2 Creating the Business Components In the *Create Business Components from Tables* wizard:

1. Complete the *Entity Objects* page as follows:
 - a. Enter *Expense%* in the *Name Filter* field.
 - b. Click **Query**. *EXPENSE_ITEMS* and *EXPENSE_REPORT* appear in the *Available* pane.
 - c. Click **Add All** to move these items to the *Selected* pane.
 - d. Click **Next**. The *Updatable View Objects* page appears (Step 2).
 - e. Click **Add All** to move *EXPENSE_ITEMS* and *EXPENSE_REPORT* to the *Selected* pane.
 - f. Click **Next** to proceed from the *Updatable View Objects* page to the *Read-Only View Objects* page (Step 3). *EXPENSE_ITEMS* and *EXPENSE_REPORT* appear in the *Available* pane.

2. Click **Add All** to move *EXPENSE_ITEMS* and *EXPENSE_REPORT* to the *Selected* pane.
3. Click **Next**. The *Application Module* page appears.
4. Accept the populated values in the *Application Module* page and click **Next** until you reach the *Summary* page.
5. Click **Finish** to complete the wizard. The business components display in the *Application Navigator* (Figure 42–12). With the business components created, you now create the published events.

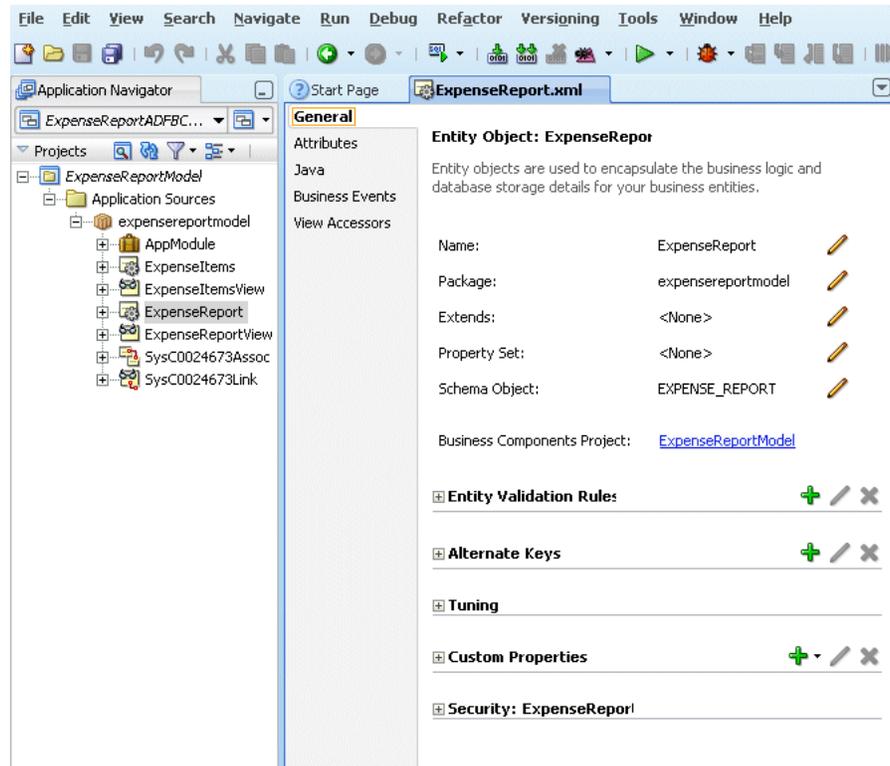
Figure 42–12 The Business Components



42.4.1.1.3 Creating Events To create events:

1. In the *Application Navigator*, open *ExpenseReport.xml* by double-clicking the *ExpenseReport* folder (located in *expensereportmodel*, as illustrated in Figure 42–12). *ExpenseReport.xml* opens, defaulting to the *General* page (Figure 42–13).

Figure 42-13 The General Page of ExpenseReport.xml



2. Select the *Business Events* page (Figure 42-14) and then click the **Add** icon. The *Edit Business Event* definition dialog appears (Figure 42-15).

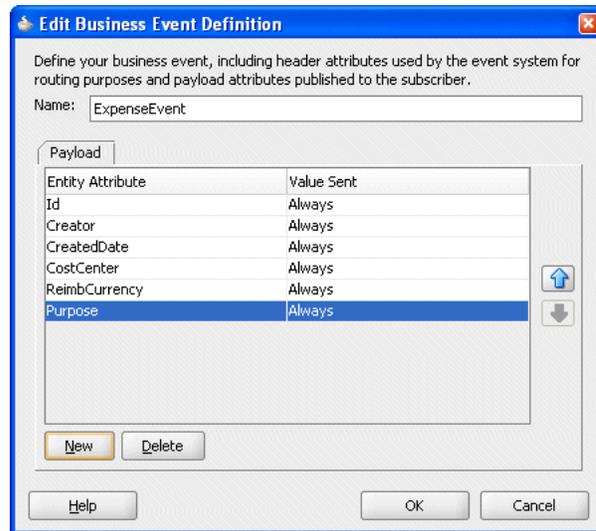
Figure 42-14 The Business Events Page of ExpenseReport.xml



3. Define the business event as follows:
 - a. Enter *ExpenseEvent* in the *Name* field.
 - b. Click **New** to add the following to the payload (Figure 42-15):
 - Id
 - Creator
 - CreatedDate
 - CostCenter
 - ReimbCurrency
 - Purpose

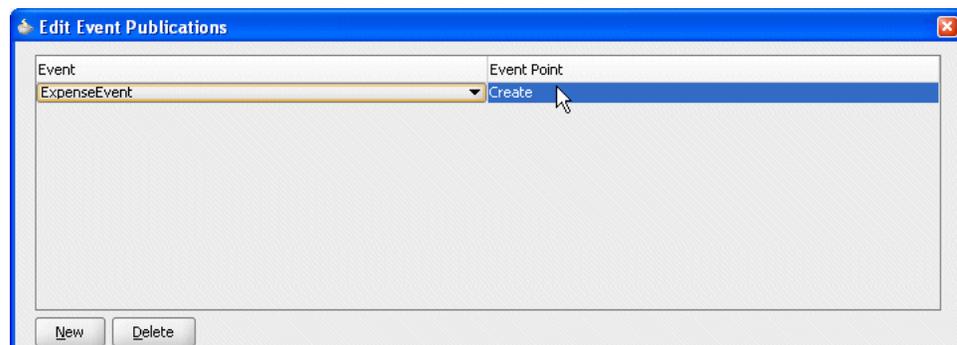
4. Accept the default *Value Sent* property, *Always*, and then Click **OK**. *ExpenseEvent* displays in the *Event Definitions* section of the *Business Events* page.

Figure 42–15 The Business Event Definition Dialog



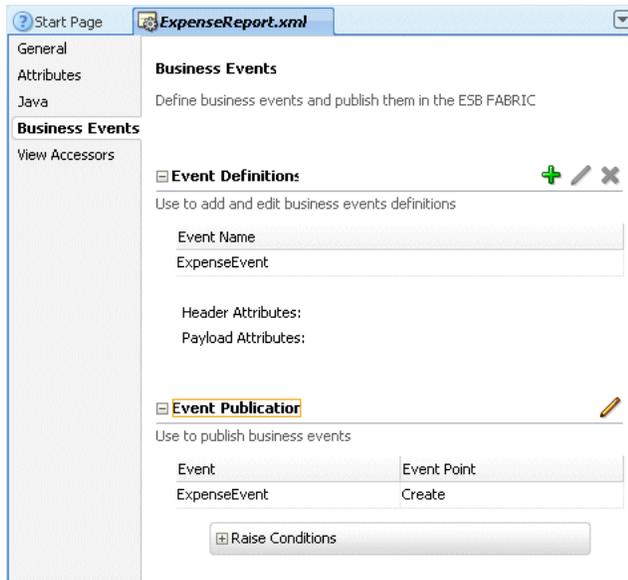
5. Define the event publication as follows:
 - a. Click the **Edit** icon (Figure 42–3) next to *Event Publication* (illustrated in Figure 42–14) to invoke the *Edit Event Publications* dialog.
 - b. In the *Edit Event Publications* dialog, click **New** to insert rows directly underneath *Event* and *Event Point*.
 - c. Double-click the newly inserted row beneath *Event*.
 - d. From the list invoked by the double-click, select *ExpenseEvent*.
 - e. Double-click the row inserted directly underneath *Event Point*.
 - f. Select *Create* from the list.

Figure 42–16 Defining the Event and Event Point



- g. Click **OK**. The *Business Events* page reappears with the event publication data populated to the *Event Publication* section (Figure 42–17).

Figure 42–17 The Business Events Page



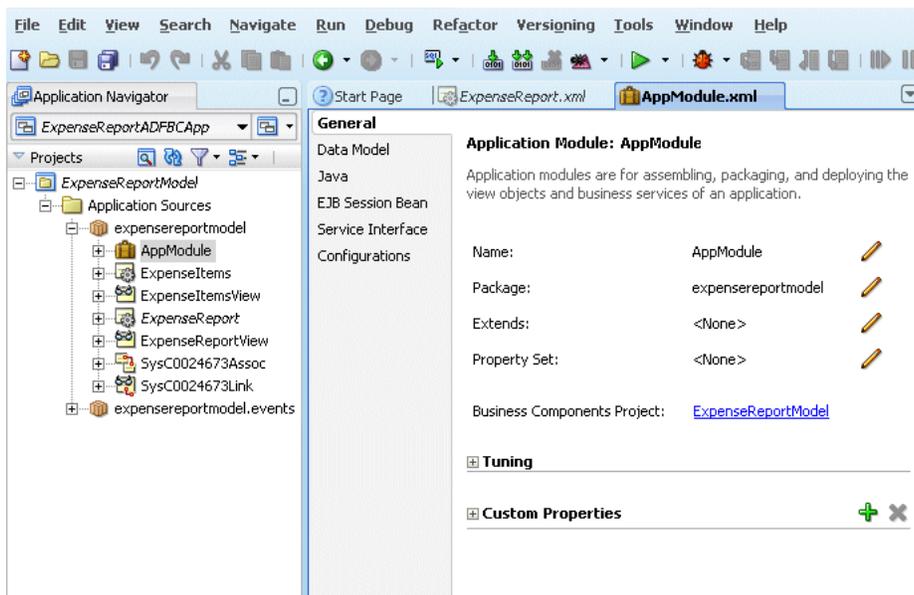
42.4.1.2 Exposing the ExpenseReport Object as a Web Service

Exposing the `update` method of the `ExpenseReport` entity object as a Web Service enables the `ExpenseReport` application’s BPEL process to call the `update` status for updating the exposed approval status based on the work flow status. For example, if a task assignee approves the expense report, then the BPEL process calls the Web Service to update the status as `Approve`.

To expose the `update` method:

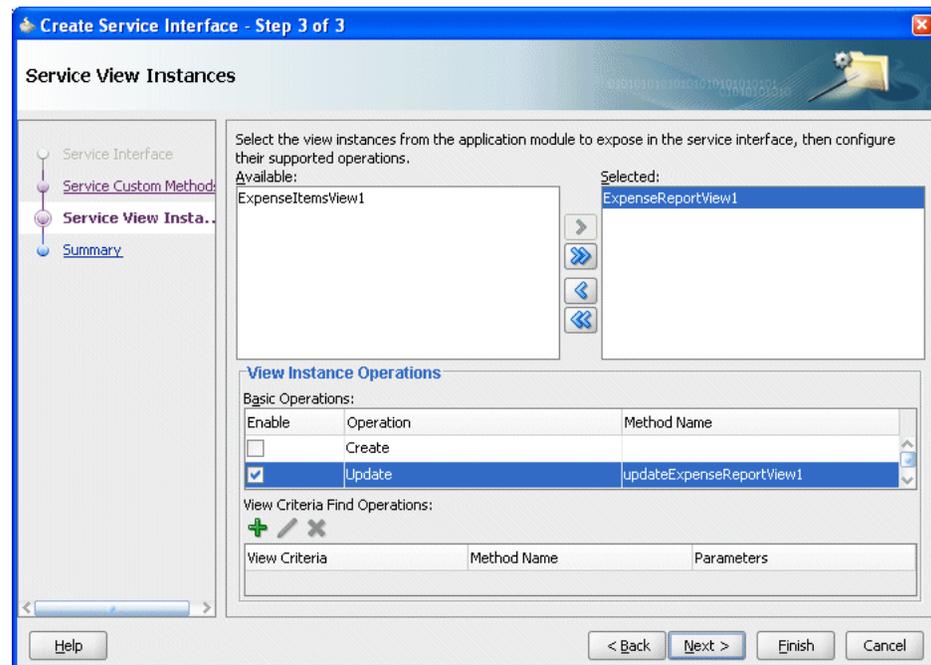
1. In the Application Navigator, Open `AppModule.xml` by double-clicking the `ExpenseReportModel` project’s `AppModule`. `AppModule.xml` appears and defaults to the `General` page (Figure 42–18).

Figure 42–18 The General Page of AppModule.xml

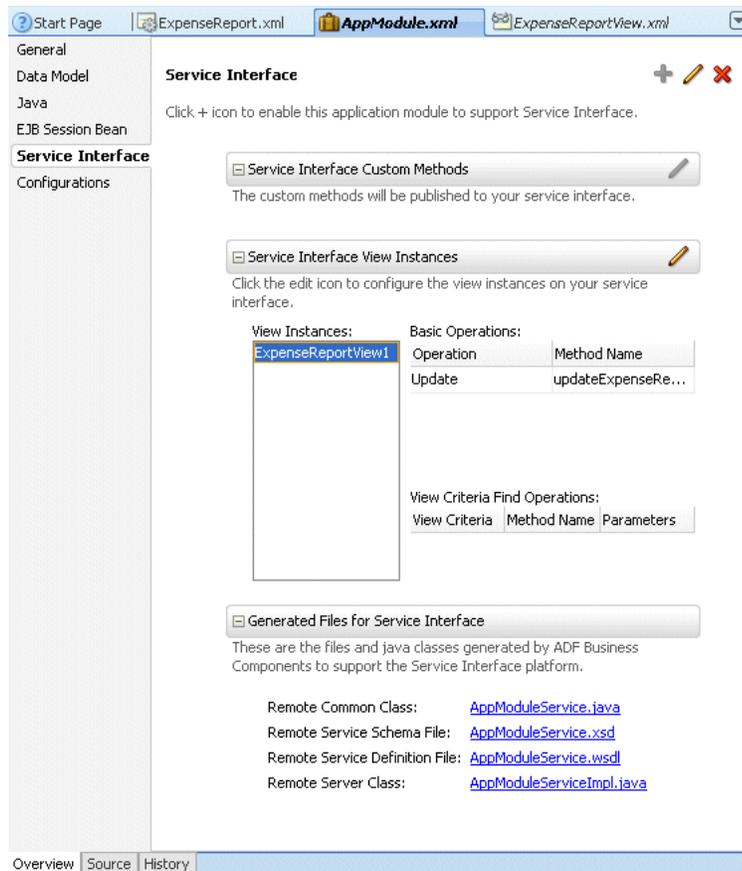


2. Click **Service Interface**. The *Service Interface* page displays.
3. Click the **Add** icon. The *Service Interface* wizard appears.
4. Accept the default values that display on the first page and click **Next** until you reach the *Service View Instance* page (Step 3).
5. Complete the *Service View Instance* page as follows:
 - a. Move *ExpenseReportView1* from the *Available* pane to the *Selected* pane.
 - b. From the *Instance View Operation* section, select **Update** operation from the *Basic Operations* table (Figure 42–19).

Figure 42–19 Creating the Updated Method



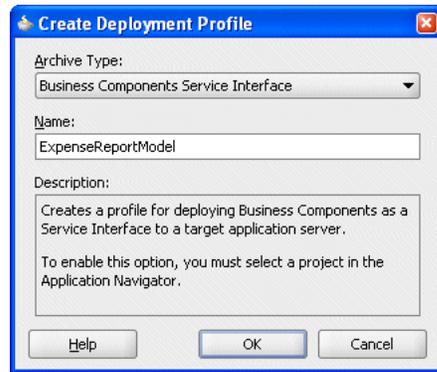
- c. Click **Finish**. *ExpenseReportView1* data displays in the *Service Interface* page (Figure 42–20).

Figure 42–20 The Service Interface Page

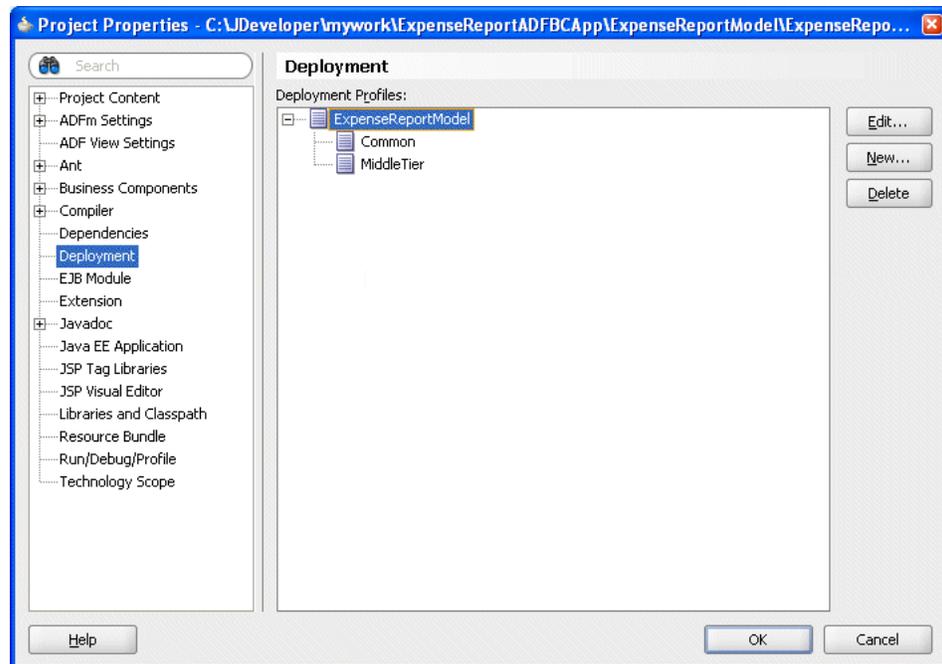
42.4.1.3 Creating the Deployment Profile for the ExpenseReportModel Project

To create the deployment profile:

1. In the Application Navigator, select and then right-click *ExpenseReportModel* to invoke the context menu.
2. Select **Project Properties**. The *Project Properties* dialog for *ExpenseReportModel* appears.
3. Select **Deployment**.
4. Click **New**. The *Create Deployment Profile* dialog appears (Figure 42–21).

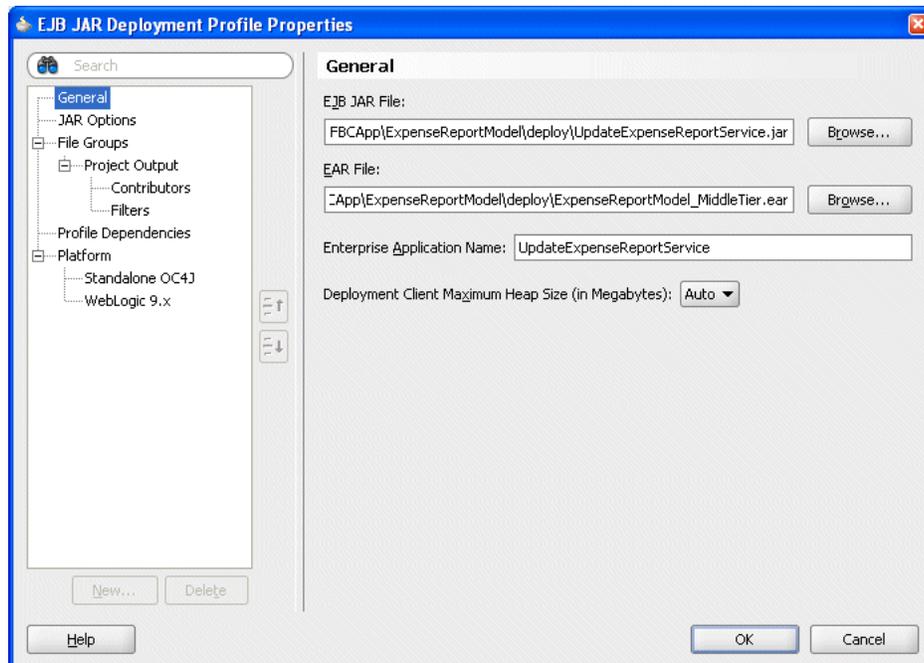
Figure 42–21 The Create New Deployment Profile Dialog

5. Select **Business Components Service Interface** from the *Archive Type* list.
6. Enter *ExpenseReportModel* in the *Name* field.
7. Click **OK**. The *ExpenseReportModel* deployment profile displays in the *Deployment Profiles* pane of the *Project Properties* dialog (Figure 42–22).

Figure 42–22 *ExpenseReportModel* Deployment Profile

8. Expand the *ExpenseReportModel* deployment profile (illustrated in Figure 42–22) and then invoke the *EJB JAR Deployment Profile Properties* dialog by either selecting **Middle Tier** and then clicking **Edit** or by double-clicking **MiddleTier**. The *EJB JAR Deployment Profile Properties* dialog appears (Figure 42–23).

Figure 42–23 Renaming the JAR File



9. Complete this dialog as follows:
 - a. In the *EJB JAR File* field, change the name of the JAR file to *UpdateExpenseReportService.jar*.
 - b. In the *Enterprise Application Name* field, enter *UpdateExpenseReportService* and then click **OK**.
10. Click **OK** to dismiss the *Project Properties* dialog.

During application deployment, add this project in the assembly, so that it will be deployed during the *ExpenseReportADFApp* application. After you deploy this application, you can test the Web Service using the following URL:

```
http://<hostname>:<port>/UpdateExpenseReportService/AppModule
Service
```

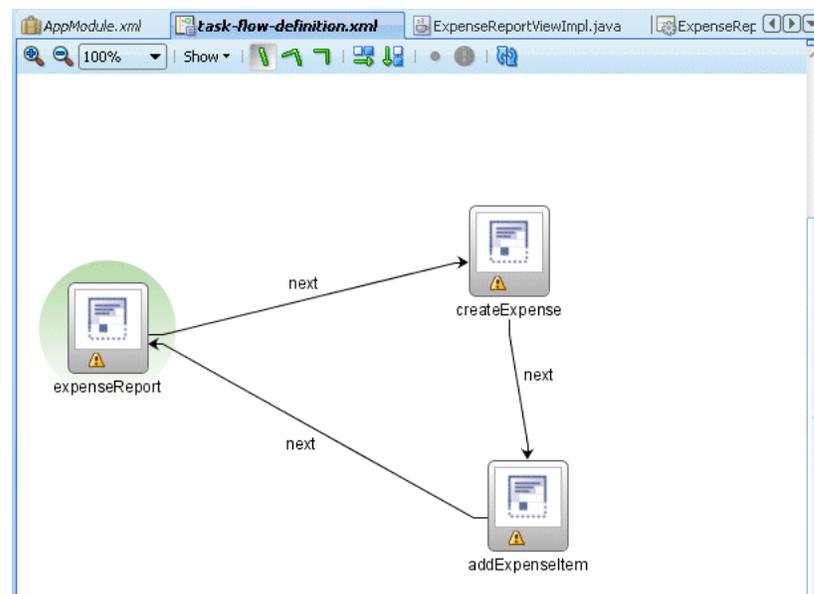
42.4.1.4 Creating the ADF View Project

To create the ADF View Project:

1. From the Application Navigator, click **New**. The *New Gallery* appears.
2. Click **Projects** (located under the *General* node) and then click **Business Components Project** in the *Items* pane and then click **OK**. The *Welcome* page of the *Create Business Components Project* wizard appears.
3. In the *Items* pane, expand **Web Tier** and then **JSF**.
4. Select **ADF TaskFlow** in the *Categories* pane. The *Create ADF Taskflow* dialog appears.
5. Click **OK**.
6. *task-flow-definition.xml* appears.
7. Drag and drop a *View* object component from the *ADF Task Flow* component palette into *task-flow-definition.xml*.

8. Name the view object *expenseReport*. This is the default view object.
9. Drag and drop two more *View* objects from the *ADF Task Flow* component palette into the *task-flow-definition.xml* file and name them as follows:
 - *createExpense*
 - *addExpenseItem*
10. Create the flow between these views as follows:
 - a. Drag a *Control Flow Case* component from the Components Palette and then drop it into *expenseReport*. Connect *expenseReport* to *createExpense*.
 - b. Label the *Control Flow Case* component as *next*.
 - c. Drag and drop a *Control Flow Case* from *expenseReport* to *addExpenseItem*.
 - d. Label the *ControlFlowCase* component as *next*.
 - e. Drag and drop a *Control Flow Case* component from *addExpenseItem* to *ExpenseReport*.
 - f. Label the *ControlFlowCase* component as *next*. [Figure 42–24](#) illustrates the view items strung together in a task flow.

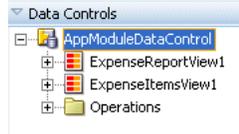
Figure 42–24 Creating the Task Flow Using the ADF Task Flow Component Palette



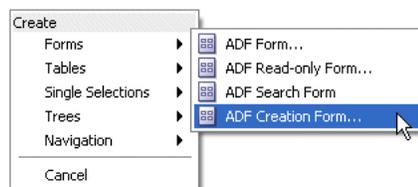
11. Create a *.jspx* page for the *expenseReport* view object by double-clicking the *expenseReport* view object. The *Create JSP* dialog appears.
12. Click **OK** to create *expenseReport.jsx*.
13. Drag a *button* component from the *Common Components* palette to *expenseReport.jsx*.
14. Double-click the button object in *expenseReport.jsx* to access the *Bind Action Property* dialog ([Figure 42–25](#)).

Figure 42–25 The Bind Action Property Dialog

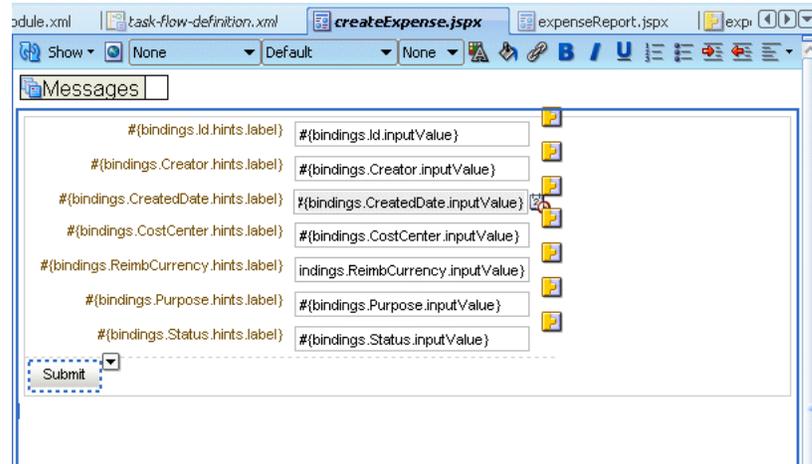
15. Select **Action Outcome** and then select **next** from the list.
16. Click **OK**.
17. Rename the button object by first placing the cursor on it and then by entering *Expense Report*.
18. Click **Enter**.
19. Create a *.jspx* page for the *createExpenseView* object by double-clicking the *createExpenseView* component. The *Create JSP* dialog appears.
20. Click **OK** to create *createExpense.jsp*.
21. If needed, expand *Data Controls* (Figure 42–26) and then drag and drop the *expenseReportView1* data control into *createExpense.jsp*.

Figure 42–26 The Data Controls Component

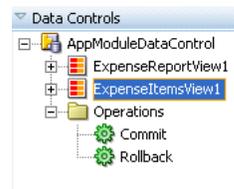
22. From the context menu that appears, select **Forms** and then the **ADF Creation Form** drop handler (Figure 42–27). The *Edit Form Fields* dialog appears (Figure 42–27).

Figure 42–27 Adding the ADF Form

23. Select **Include Submit** and then click **OK**. The bindings display in *createExpense.jsp* (Figure 42–28).

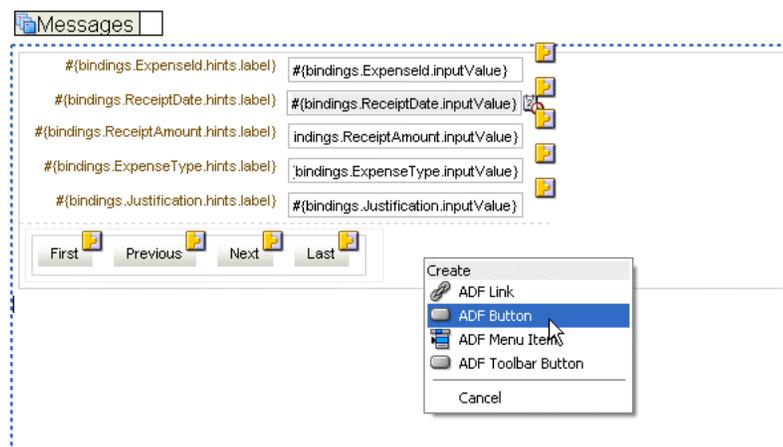
Figure 42–28 Bindings

24. Double-click the **Submit** button. The *Bind Action Property* dialog appears (Figure 42–25).
25. Select **Action Outcome** and then select **next** from the list.
26. Click **OK**.
27. In *task-flow-definition.xml*, double-click the *addExpenseItem* view object. The *Create JSP* dialog appears. Click **OK** to create *addExpenseItem.jspx*.
28. If needed, expand *Data Controls* and then drag and drop the *ExpenseItemView2* data control component into *addExpenseItem.jspx*.
29. From the context menu that appears (Figure 42–27), select **Forms** and then select the **ADF Form** drop handler. The *Edit Form fields* dialog appears.
30. To enable the commit functionality that allows users to save changes, add a *Commit* button to *addExpenseItem.jspx*. To do this, first select **Include Navigation Controls** and then click **OK**.
31. If needed, expand the *Operations* folder of *Data Controls* (Figure 42–29).

Figure 42–29 The Operations

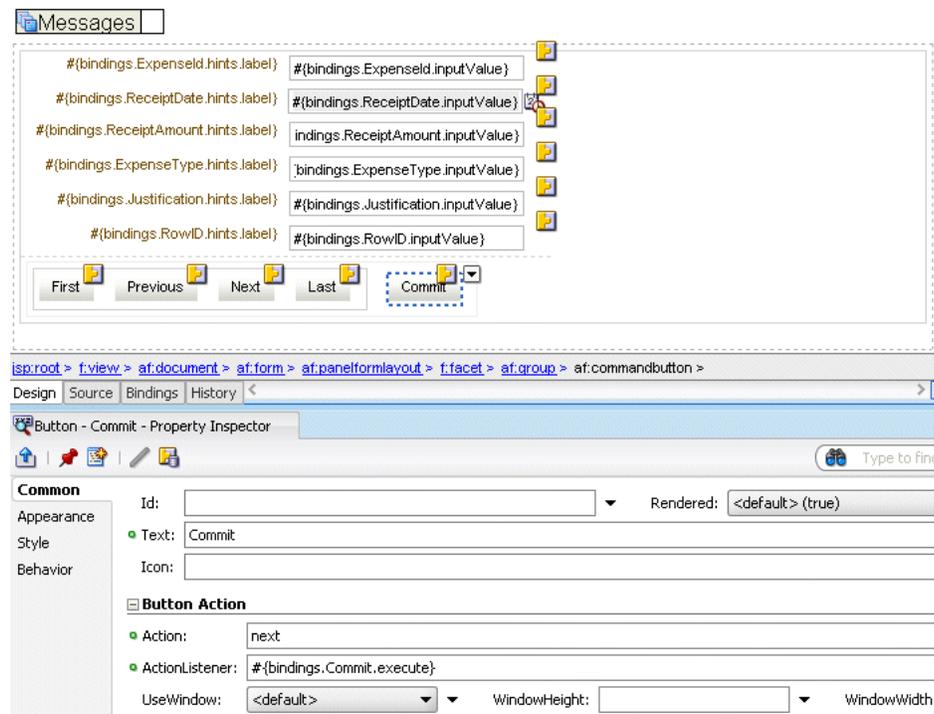
32. Drag and drop the *Commit* operation into *addExpenseItem.jspx* next to the navigation components (*First*, *Previous*, *Next*, and *Last*, illustrated in Figure 42–30).
33. From the context menu that appears, select **ADF Button** (illustrated in Figure 42–30). The *Commit* button appears (Figure 42–31).

Figure 42–30 Adding the Commit Operation



34. In the Property Inspector for the *Commit* button (called *Button-Commit-PropertyInspector* in [Figure 42–31](#)), expand *Button Action* (if needed) and select **next** from *Action* list.

Figure 42–31 Editing the Properties for the Commit Button



35. Click **Save All**.

42.4.1.5 Creating the Deployment Profile for the ADF BC Application

To deploy the application:

1. Click **File** and then **New**. The *New Gallery* appears.
2. Expand the *General* node (if needed) and then select **Deployment Descriptors**.
3. Select **OC4J Deployment Descriptor Wizard**.

4. Click **OK**. The *Create OC4J Deployment Descriptor Wizard* appears and opens to the *Select Descriptors* page.
5. Complete the wizard as follows:
 - a. In the *Select Descriptors* page, select **orion-application.xml** and then click **Next**. The *Select Version* page appears.
 - b. Select (or accept) **10.0** and then click **Finish** to create *orion-application.xml*.
6. Right-click *ExpenseReportModel* and then select *Project Properties* from the context menu.
7. Select **Compiler**. Accept the default values in the *Compiler* pane and then click **OK**.
8. Click **File** and then **New**. The *New Gallery* appears.
9. Expand the *General* node (if needed) and select **Deployment Profiles**.
10. Select **WAR file** from the *Items* pane.
11. Click **OK**. The *Create Deployment File – WAR File* dialog appears.
12. Enter *ExpenseReportDeploymentProfile* in the *Deployment File Name* field.
13. Click **OK**. The *WAR Deployment Profile Properties* appears.
14. Click **OK**.
15. In the Application Navigator, right-click the *ExpenseReportModel* project and then select **Deploy** from the context menu.

42.4.2 Creating the SOA Composite Application (ExpenseReportComposite)

To create the SOA application and its project files:

1. Click **File** and then click **New**. The *New Gallery* appears.
2. In the *Items* pane, select **Application** and then click **OK**. The *Create Application Dialog* appears.

Note: You must select **No Template [All Technologies]** (the default setting) from the *Application Template* list.

3. Enter *ExpenseReportComposite* in the *Application Name* field. If needed, enter an alternate location for the application in the *Directory* field. Otherwise, accept the default location (*C:\JDeveloper\mywork\ExpenseReportComposite*) and click **OK** to create the application. The *Create Project* dialog appears.
4. Click **Cancel** to dismiss the *Create Project* dialog.

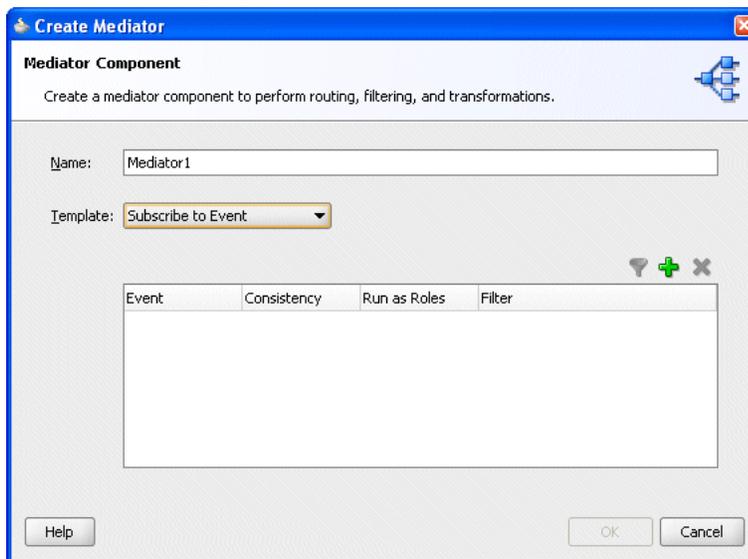
42.4.2.1 Creating the Mediator

To create the mediator:

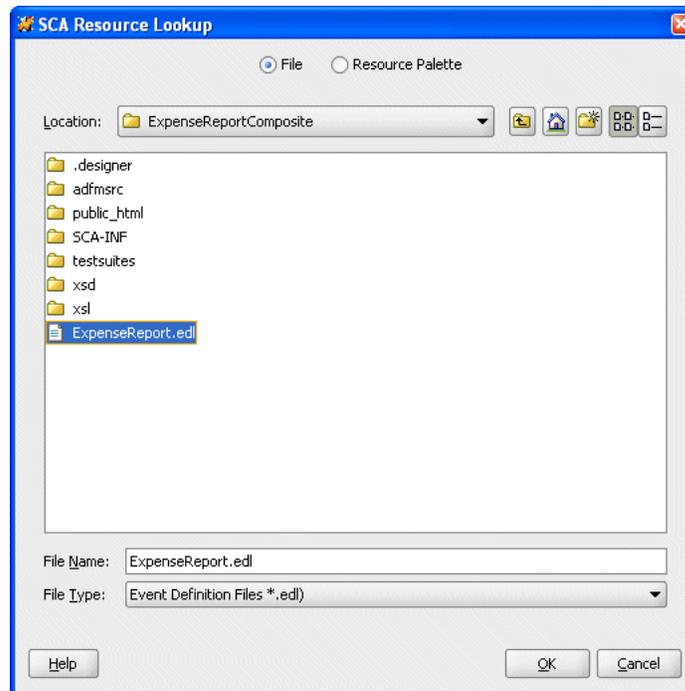
1. Click **File** and then **New**. The *New Gallery* appears.
2. Click **Project** and then click **SOA** in the *Items* pane. The *Create SOA Project* dialog appears.
3. Complete the *SOA Project Dialog* as follows:

- a. Enter *ExpenseReport_Prj* in the *Project Name* field. If needed, enter an alternate location for the application in the *Directory* field. Otherwise, accept the default location (*C:\JDeveloper\mywork\ExpenseReport\ExpenseReport_Prj*).
- b. In the *Composite Template* field, select **Composite with Mediator**.
- c. Click **OK** to create the project file (*.jpr*). The *Create Mediator* dialog appears (Figure 42–32).

Figure 42–32 The Create Mediator Dialog



4. Complete the *Create Mediator* dialog as follows:
 - a. Enter *ExpenseReportMediator* in the *Name* field.
 - b. Select **Subscribe to Event** from the *Options* list.
 - c. Click the **Add** icon (Figure 42–6) to select the event type. The *Event Chooser* dialog appears (Figure 42–34).
 - d. Click the flashlight icon. The *SCA Resource Lookup* appears (Figure 42–33).
 - e. Navigate to the location of *ExpenseReport.edl* by selecting **File** and then traverse to the *ExpenseReport.edl* file, located in the *ExpenseReportComposite* directory.

Figure 42–33 The SCA Resource Lookup Dialog

- f. Click **OK**. The *Event Chooser* reappears (Figure 42–34) with the location of *ExpenseReport.edl* in the *Definition* field, *ExpenseEvent* in the *Event* field, and the schema definition of *ExpenseEvent* in the *Type* field.

Figure 42–34 The Event Chooser Dialog

5. Click **OK**. The *ExpenseReportMediator* component appears in the SOA Composite Editor (the *composite.xml* file).

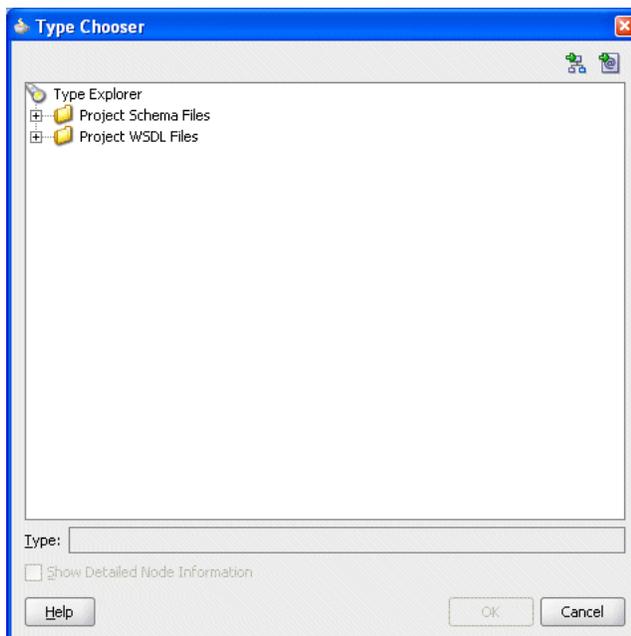
Tip: If the *Copy File* dialog appears, click **Yes** to create a local copy of *ExpenseReport.edl*.

Figure 42–35 The ExpenseReport Mediator Component

42.4.2.2 Creating the BPEL Process

To create the BPEL Process:

1. Drag the BPEL Process from the Component Palette (SOA must be selected) and drop it into the SOA.Composite Editor. The *Create BPEL Process* dialog appears (Figure 42–39).
2. Enter *ExpenseReport* in the *Name* field.
3. Select **Asynchronous BPEL Process** as the template to create a two-way process.
4. Select **Expose as Composite Service** to create an inbound SOAP service-binding component that is connected to the ExpenseReport BPEL process.
5. Define the *Input Element* and *Output Element* fields by importing the *Expense.xsd* file (located in *ExpenseReportCompositeApp\ExpenseReportComposite\xsd*) using the *Type Chooser* and *SCA Resource Lookup* dialogs (Figure 42–36 and Figure 42–33, respectively).
 - a. To import the schema, first click the flashlight icon to access the *Type Chooser* dialog.

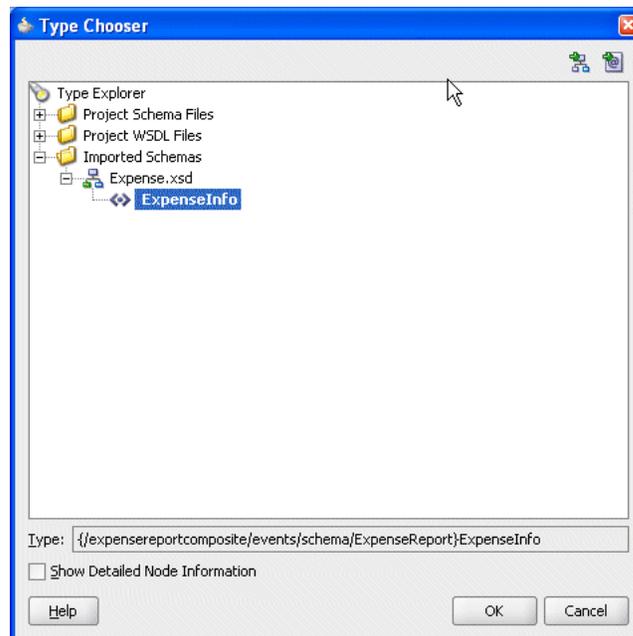
Figure 42–36 The Type Chooser Dialog

- b. Click the *Import Schema* icon (Figure 42–37).

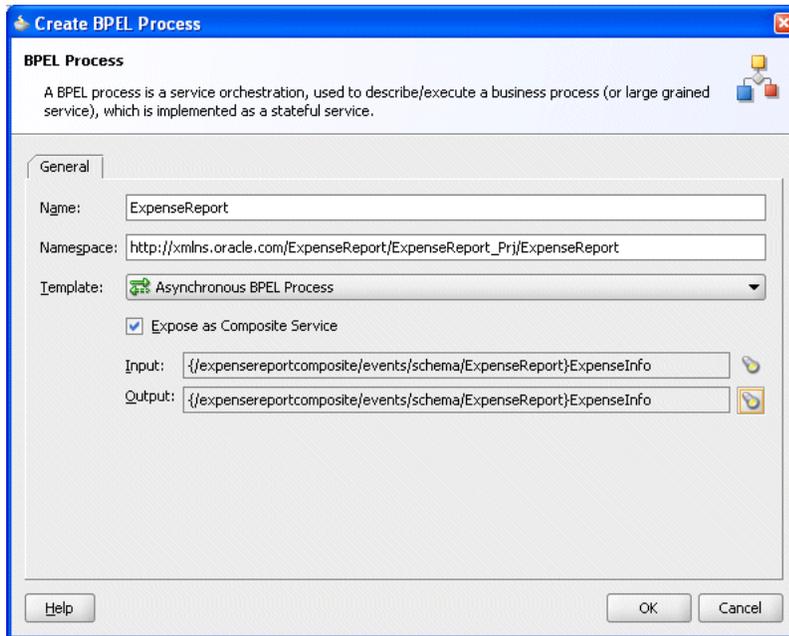
Figure 42–37 The Import Schema Icon

The *Import Schema* dialog appears.

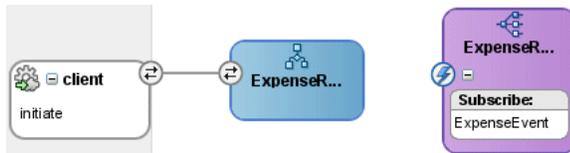
- c. Click the flashlight icon to access the *SCA Resource Lookup* dialog.
- d. With **File** selected, navigate to the *Expense.xsd* file (located in *ExpenseReportCompositeApp\ExpenseReportComposite\xsd*).
- e. Click **OK** in the *Import Schema* dialog. Click **Yes** if you are prompted to copy the file to JDeveloper.
- f. In the *Type Chooser* dialog, expand *Imported Schemas* and then select **ExpenseInfo** (Figure 42–38).
- g. Click **OK**.

Figure 42–38 The Type Chooser Dialog

6. Repeat these steps for both the *Input* and *Output* elements. Click **OK** when complete (Figure 42–39).

Figure 42–39 The Create BPEL Process Dialog (Completed)

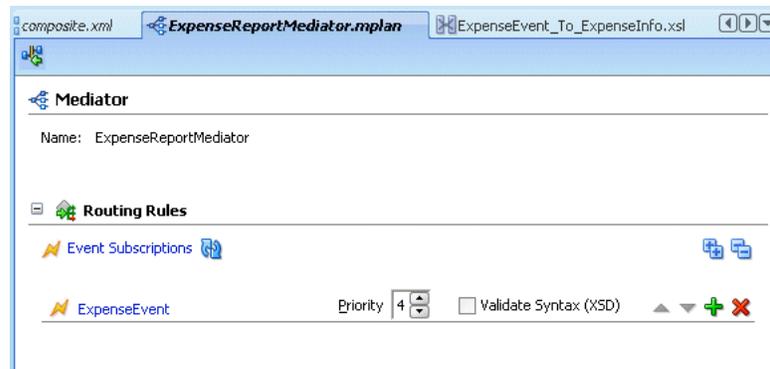
The ExpenseReport BPEL process appears in the SOA Composite Editor. It is wired to a client component, but is independent of the ExpenseReport mediator component (Figure 42–40).

Figure 42–40 The ExpenseReport BPEL Process in the SOA Composite Editor

42.4.2.3 Wiring the Mediator to the BPEL Process

Wiring enables the *ExpenseReportMediator* component to communicate with the Expense Report BPEL process so that the event can initiate the Expense Report BPEL process.

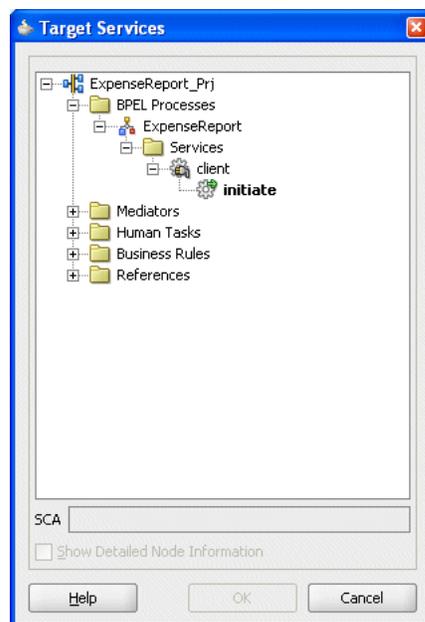
1. In the SOA Composite Editor, double-click *ExpenseReportMediator* component (Figure 42–35). The mediator component editor appears (Figure 42–41).

Figure 42–41 The Mediator Editor

2. If needed, expand the *Routing Rules* section (illustrated in [Figure 42–41](#)).
3. Click the **Add** icon in the *Routing Rules* section. The *Target Type* dialog appears.

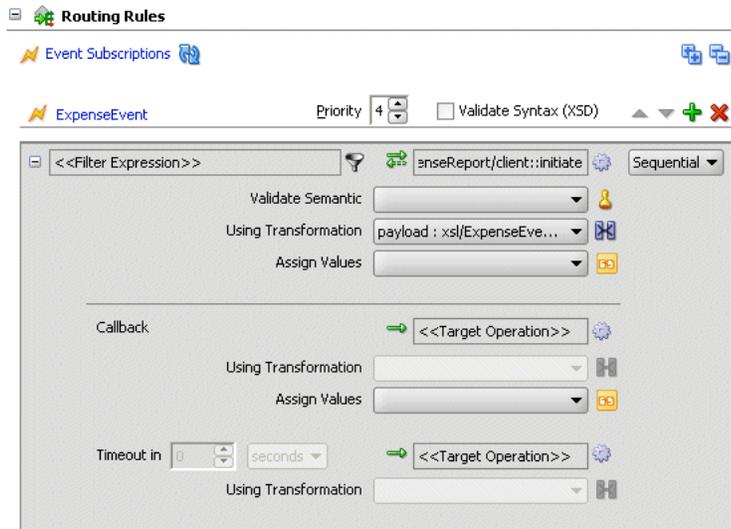
Figure 42–42 The Target Type Dialog

4. Click **Service**. The *Target Services* dialog appears.
5. Expand the *BPEL Processes* folder and then select the **initiate** operation of the *ExpenseReport* BPEL process ([Figure 42–43](#)).

Figure 42–43 The Target Services Dialog

- Click **OK**. The *Routing Rules* panel of the mediator component editor is populated with the *ExpenseReport* data, with *xsl/ExpenseEvent_To_ExpenseInfo.xsl* displaying in the *Using Transformation* field (Figure 42–44).

Figure 42–44 *ExpenseReport* Data Retrieved by the *initiate* Operation



42.4.2.4 Transforming the Data

Oracle JDeveloper provides an XSLT Data Mapper tool that enables you to specify a mapper file (that is, an XSL file) to transform data from one XML schema to another. This enables data interchange among applications using different schema.

To create a mapping:

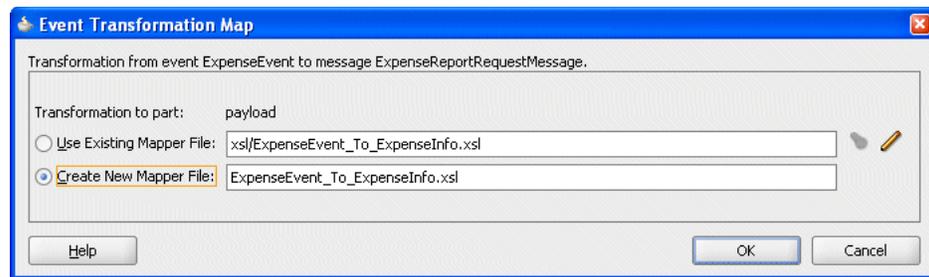
- In mediator component editor, click the mapping icon (Figure 42–45) that is adjacent to the *Using Transformation* field in the *Filter Expression* section (Figure 42–44).

Figure 42–45 *The Mapping Icon*

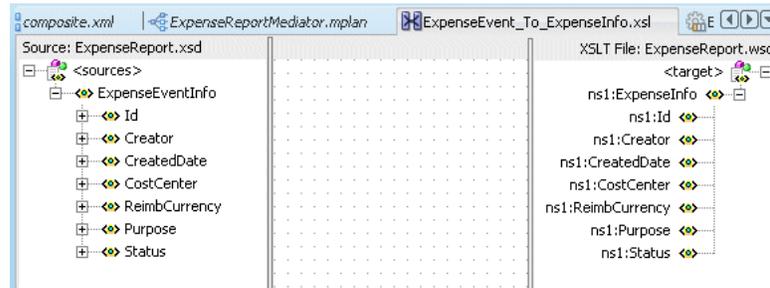


The *Event Transformation Map* dialog appears (Figure 42–46), with *xsl/ExpenseEvent_To_ExpenseInfo.xsl* displaying in the field corresponding to the *Use Existing Mapper File* option.

- Select **Create New Mapper File**. *ExpenseEvent_To_ExpenseInfo.xsl* displays in the field (Figure 42–46).

Figure 42–46 The Event Transformation Map Dialog

3. Click **OK**. A tab called *ExpenseEvent_to_ExpenseInfo.xsl* is added to the Oracle JDeveloper console (Figure 42–47). This tab enables you to graphically create a document transformation file to convert the structure of the file data to a canonical data structure. There are no links from the source elements of *ExpenseReport.xsd* (located in the left pane) to their counterparts in the target (*ExpenseReport.wsdl*, located in the right pane.)

Figure 42–47 ExpenseEvent_to_ExpenseInfo.xsl

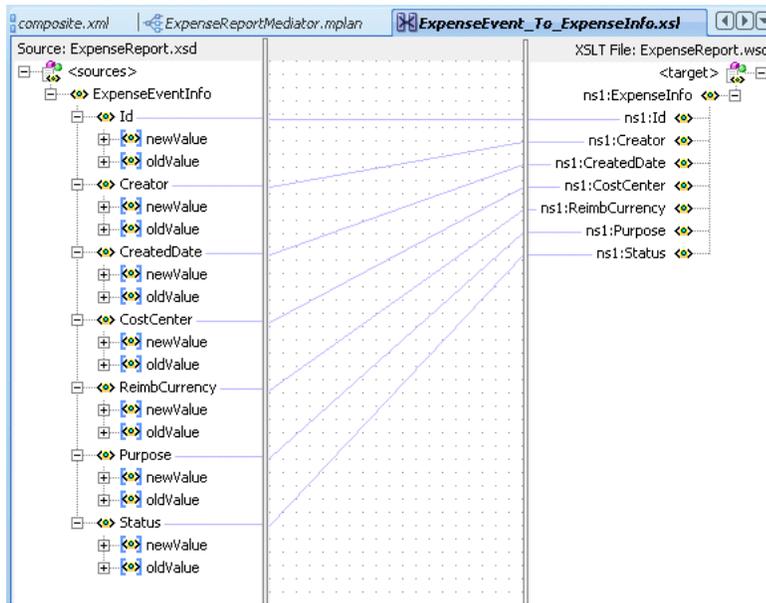
4. Create links between the matching elements by dragging and dropping each of the source elements to the corresponding target elements. Create the following links described in Table 42–2.

Table 42–2 Creating Links Using ExpenseReport.xsd Elements

From ExpenseReport.xsd...	...To ExpenseReport.wsdl
Id	ns1:Id
Creator	ns1:Creator
CreatedDate	ns1:CreatedDate
CostCenter	ns1:CostCenter
ReimbCurrency	ns1:ReimbCurrency
Purpose	ns1:Purpose
Status	ns1:Status

5. Click **Save**.

Figure 42–48 Matching the Source to the Target Elements



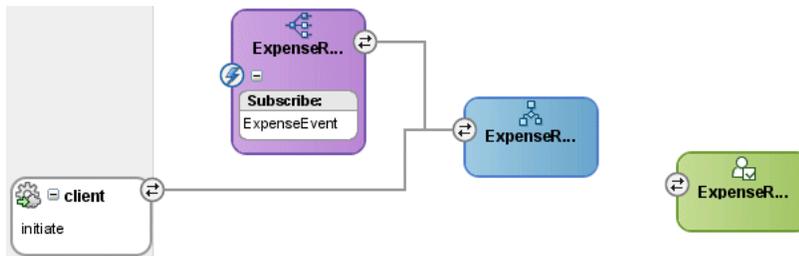
42.4.2.5 Creating the Human Workflow

To add a human task

1. Drag the *Human Task* component from the SOA Components palette to the SOA Composite Editor. The *Create Human Task* dialog appears.
2. Define the task as follows:
 - a. Enter *ExpenseReportTask* in the *Human Task Definition Name* field.
 - b. Click **OK**. The *ExpenseReportTask* component appears in the SOA Composite Editor (Figure 42–49).

Note: Because this human task is explicitly created for a BPEL process, the **Expose as Composite Service** option is not selected.

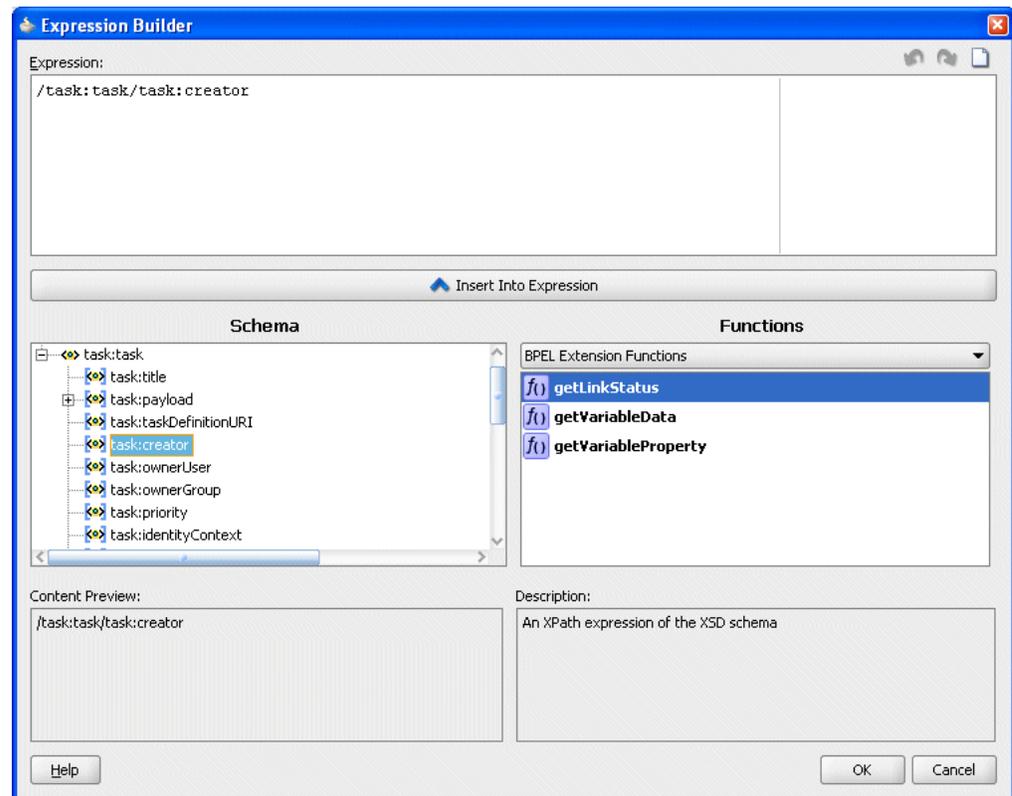
Figure 42–49 The ExpenseReportTask Human Task Component



3. As illustrated in Figure 42–49, the ExpenseReport BPEL process component and ExpenseReportTask component are not wired. Use the Human Task Editor to link these components. To access the Human Task editor, double-click the *ExpenseReportTask* component.
4. Enter *Expense Report Filed By* in the *Title* field.

5. To specify the task owner dynamically (as opposed a static or dedicated user from a directory configured for Oracle BPEL Process Manager), select **XPATH** from the list next to the *User* field and then click the adjacent note pad icon to access the *Expression Builder* dialog (Figure 42–50).
 - a. Expand **task:task** in the *Schema* pane.
 - b. Select **task:creator**.
 - c. Click **Insert Into Expression** and then click **OK**.
/task:task/task:creator appears in the *User* field of the *Human Task Editor*.

Figure 42–50 The Expression Builder Dialog

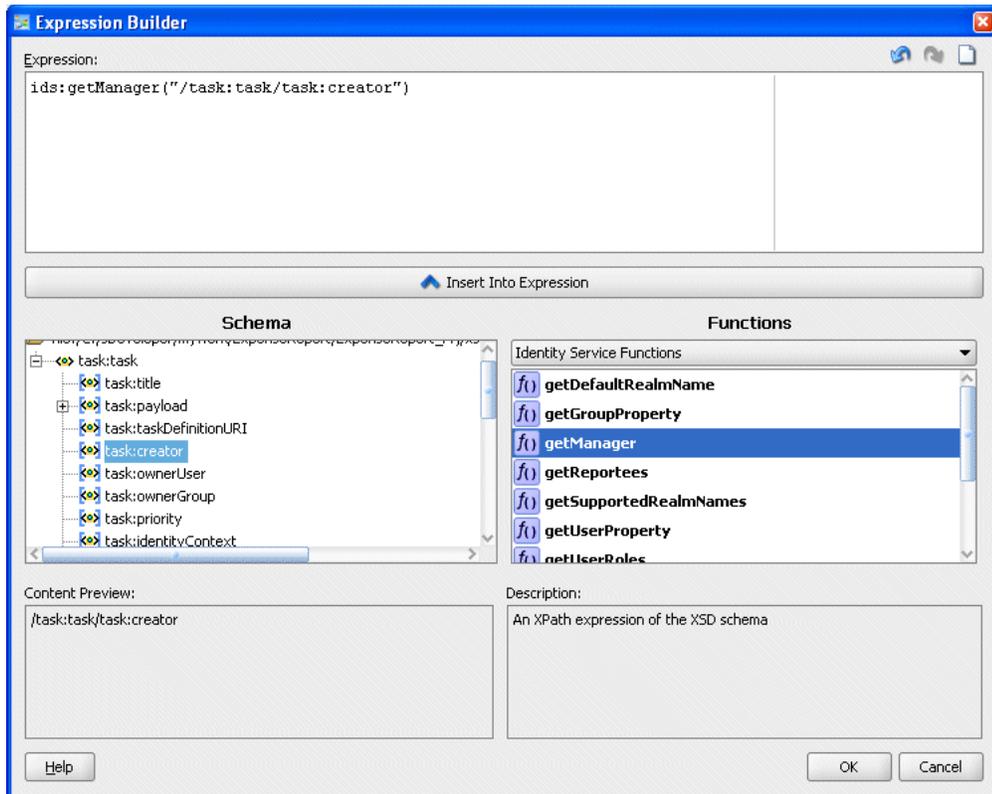


6. Specify the payload data structure as follows:
 - a. Click the **Add** icon in the *Parameters* section to invoke the *Add Task Parameter* dialog.
 - b. Enter *Creator* in the *Name* field and then click **OK**.
 - c. Repeat these steps to create a parameter called *ExpenseID*.
7. Create the management chain approval type based on the creator of the task by first clicking the **Add** icon in the *Assignment and Routing* policy section to invoke the *Add Participant Type* dialog.
8. In the *Add Participant Type* dialog, complete the *General* section as follows:
 - a. Select **Management Chain** from the list.
 - b. Select **By expression**.

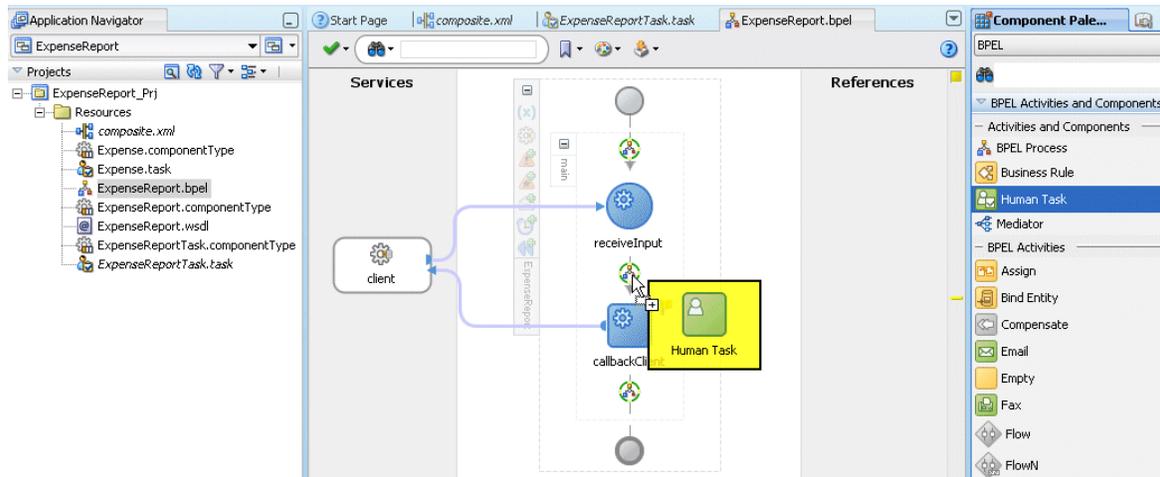
- c. In the *Dynamic User XPATH* field, click the note pad icon to access the *Expression Builder* dialog.
- d. Build the following expression:


```
ids:getManager("/task:task/task:creator")
```

Figure 42–51 *Creating the getManager Expression*



9. Click **OK** to exit the *Expression Builder* dialog.
10. Complete the *Number of Approvers* section as follows:
 - a. *Maximum Number of Chain Levels Up*: Select **By Number** and then enter 2.
 - b. *Highest Title of Approver*: Select **By Title** and then select *Director*.
11. Click **OK** to exit the *Add Participant Type* dialog.
12. Enable participants to approve or reject the expense report through e-mail as follows:
 - a. Click the **Add** icon in the *Notification Settings* section.
 - b. Select **Make email messages actionable**.
13. Create a new task as follows:
 - a. Open the BPEL flow, *ExpenseReport.bpel*, either by clicking the *ExpenseReport.bpel* tab or by selecting it from the *Resources* file in the Application Navigator.
 - b. Drag a *Human Task* from the BPEL component palette and insert it below the *ReceiveInput* component (Figure 42–52).

Figure 42–52 Adding a Human Task Component to the BPEL Flow

The *Add a Human Task* dialog appears. *ExpenseReportTask* displays in the *Task Definition* list and *Creator* and *ExpenseID* display in the *Task Parameters* column (Figure 42–53).

Figure 42–53 The Add a Human Task Dialog

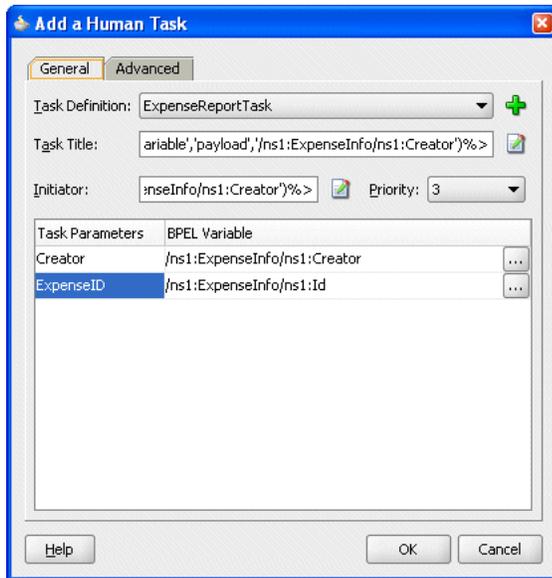
- c. To insert a dynamic task title, click the note pad icon next to the *Task Title* field. The *Expression Builder* dialog appears.
- d. In the BPEL Variables pane, navigate to the payload of the *inputVariable* and then **select ns1:creator**.
- e. Click **Insert Into Expression** and then click **OK**.
- f. Repeat these steps to select **ns1:creator** as the value for the *Initiator* field.
- g. To define the *Creator* and *ExpenseID* task parameters, click the adjacent search icon (Figure 42–54).

Figure 42–54 The Search Icon



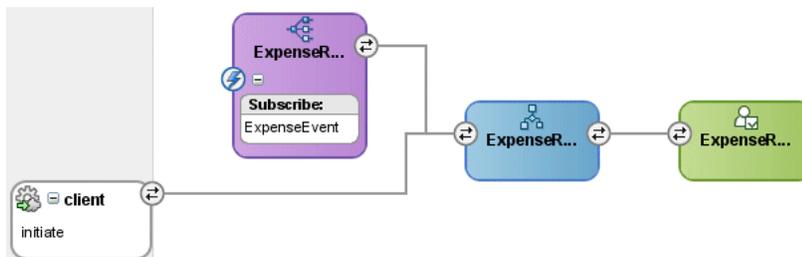
- h. In the *Task Parameters* dialog that appears, expand the *Variables* navigation tree and select the appropriate task variables. For *Creator*, select **ns1:creator**. For *ExpenseID*, select **ns1:ID**.
 - i. Click **OK**.
14. Once you complete the *Add a Human Task* dialog (Figure 42–55), click **OK**.

Figure 42–55 The Completed Add a Human Task Dialog



Once the human task is completed, the *ExpenseReportTask* component is wired to the *ExpenseReport* BPEL workflow object in the SOA Composite Editor (Figure 42–56).

Figure 42–56 ExpenseReportTask Wired to the BPEL Process



42.4.2.6 Deploying the Composite Application

See Section 42.2.1, "Deploying the SOA Composite and Task Flow Applications (ExpenseReportCompositeApp)"

42.4.3 Creating the Task Flow Application (ExpenseAppHumanTaskFlow)

To create the task flow application:

1. Click **File** and then click **New**. The **New Gallery** appears.
2. From the **Categories** pane, expand **General** and then select **Applications**.
3. In the **Items** pane, select **Application** and then click **OK**. The *Create Application Dialog* appears.

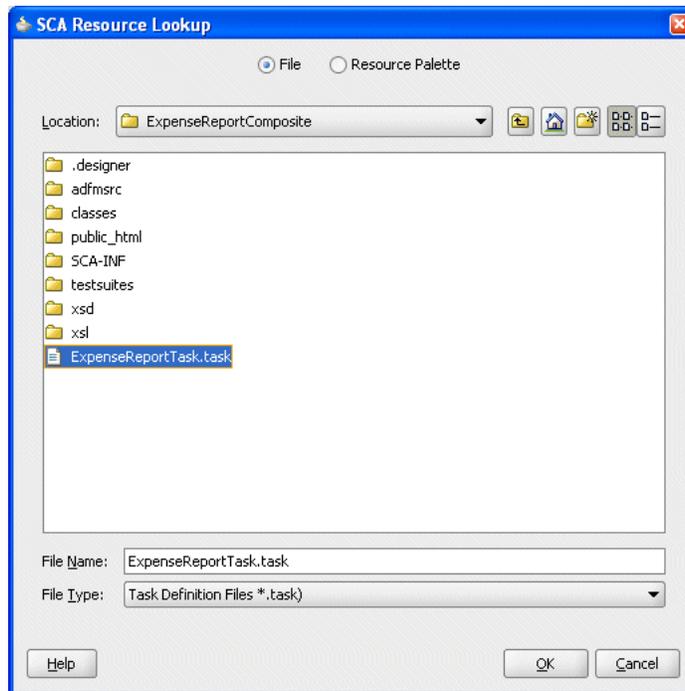
Note: **No Template [All Technologies]** (the default setting) must be selected from the Application Template list.

4. Enter *ExpenseAppHumanTaskFlowApp* in the *Application Name* field. If needed, enter an alternate location for the application in the *Directory* field. Otherwise, accept the default location (*C:\JDeveloper\mywork\ExpenseAppHumanTaskFlowApp*) and click **OK** to create the application. The *Create Project* dialog appears.
5. Click **Cancel** to dismiss the *Create Project* dialog. *ExpenseAppHumanTaskFlowApp* appears in the Application Navigator.

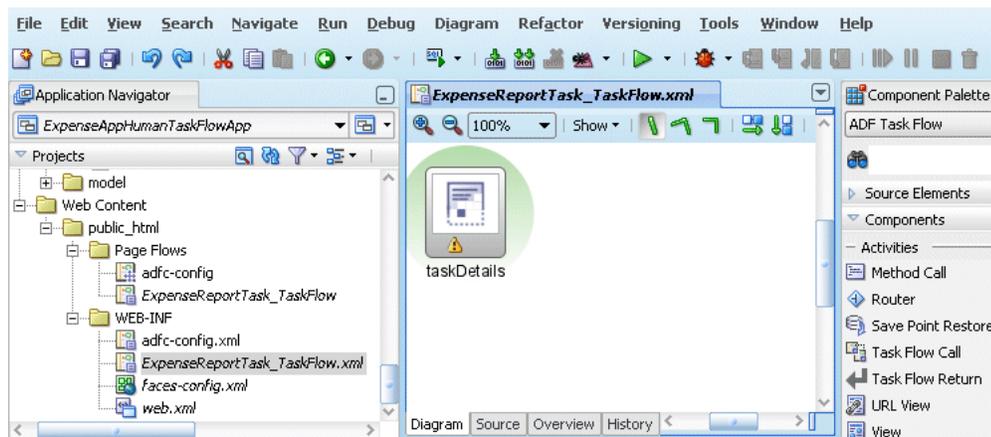
42.4.3.1 Creating the Task Flow Project

To create the project:

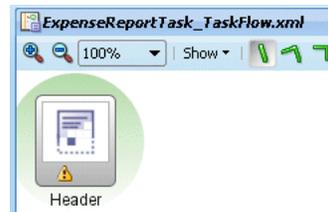
1. Right-click *ExpenseAppHumanTaskFlowApp* in the Application Navigator.
2. Select **New Project**. The **New Gallery** Appears.
3. If needed, select **Projects** in the *Categories* pane and then **Empty** from the *Items* pane.
4. Click **OK**. The *New Projects* dialog appears.
5. Enter *ExpenseAppHumanTaskFlowApp* in the *Project Name* field.
6. Click **OK**. *ExpenseAppHumanTaskFlowApp* appears in the Applications Navigator.
7. Right-click the *ExpenseAppHumanTaskFlowApp* project and then select **New** from the context menu. The *New Gallery* appears.
8. In the *Categories* pane, expand the *Web Tier* node and then select **JSF**.
9. In the *Items* pane, select **ADF Task Flow Based on Human Task** and then select **OK**. The *SCA Resource Lookup* appears (Figure 42-57).
10. Select **File** and then navigate to **ExpenseReportTask.task** (located in the *ExpenseReportComposite*) directory.

Figure 42–57 The SCA Resource Lookup

11. Click **OK**. The *Create ADF Task Flow* dialog appears.
12. Click **OK**. The task flow design page entitled *ExpenseReportTask_TaskFlow.xml* appears with a default *Task View* object called *taskDetails* (Figure 42–58).

Figure 42–58 ExpenseReportTask_TaskFlow.xml

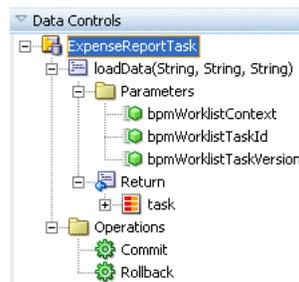
13. Rename the *taskDetails* object *Header* (accomplished by clicking the object's caption).

Figure 42–59 The Header Task View Object (Formerly taskDetails)

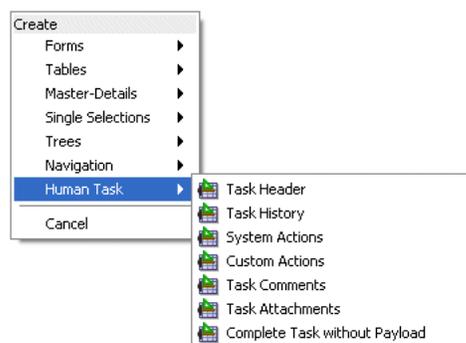
42.4.3.1.1 Populating the Header Task Flow Object

To populate the Header object:

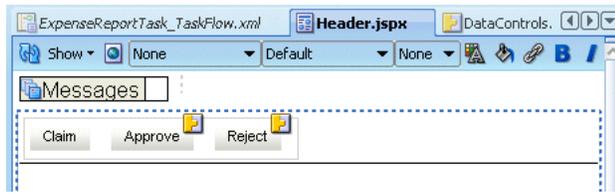
1. Double-click the *Header* Task View object. The *Create JSF Page* dialog appears.
2. Click **OK**. *Header.jspx* appears.
3. In the Applications Navigator's Data Controls pane, expand *ExpenseReportTask* to expose the *task* node.

Figure 42–60 ExpenseReportTask

4. Drag and drop the task node into *Header.jspx*. In the context menu that appears, select **Human Task**. The *Task Handler* context menu, which includes the *Custom Actions* drop handler appears (Figure 42–61).

Figure 42–61 The Human Task Context Menu

5. Select the **Custom Actions** drop handler.
6. The *Edit Action Binding* dialog appears.
7. Click **OK**. The custom action panel group layout appears, which includes the *Claim*, *Approve* and *Reject* buttons (Figure 42–62).

Figure 42–62 The Custom Action Panel

Tip: Perform either of the following if the elements do not render properly in the JSPX window:

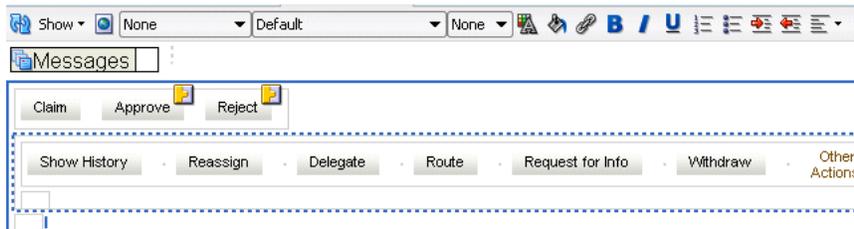
- Close JSPX page by closing *Header.jspx* and then double-click *Header* component (Figure 42–59) to reopen it. Elements, such as the *Claim*, *Approve*, and *Reject* buttons, will render properly.
- Click the *Source* tab for the JSPX page (*Header.jspx*) and insert a blank space anywhere within the document. Return to the *Design* view.

8. Click **Save All**.

9. Drag and drop the *task* icon again from Data Controls into the JSPX window.

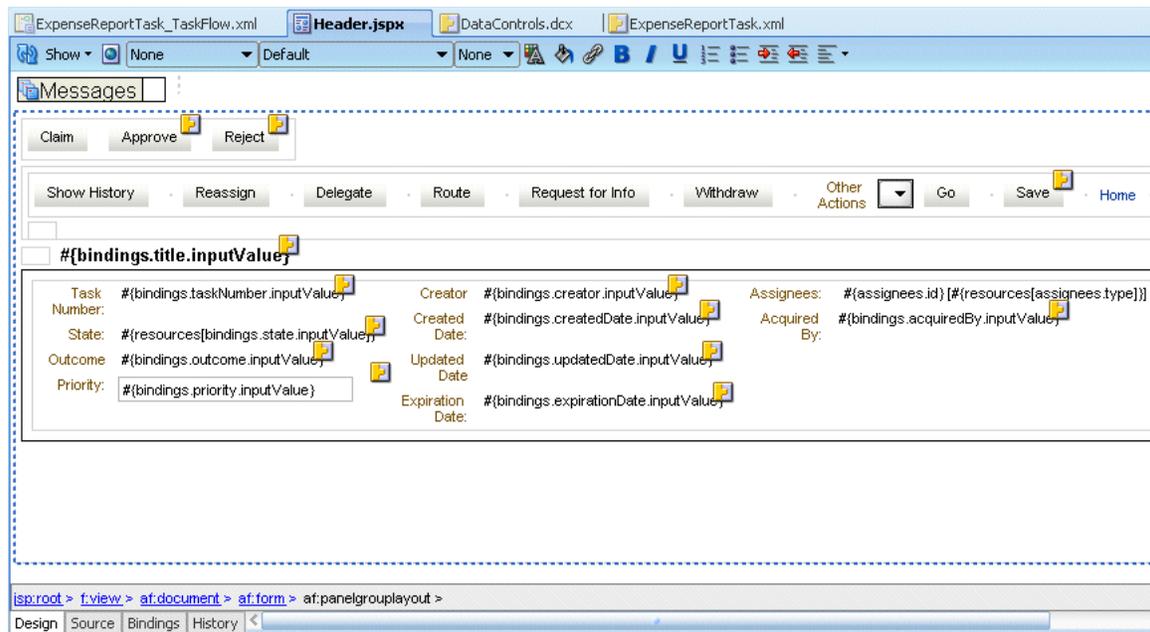
10. Drag and drop another *task* icon from Data Controls beneath the *System Actions* panel.

From the context menu (Figure 42–61), select **Human Task** and then the **System Actions** drop handler. The *Edit Action Binding* dialog appears. Click **OK**. The *System Actions* panel, that contains a second group of buttons, one that includes *Show History*, *Reassign*, and *Delegate*, appears (Figure 42–63).

Figure 42–63 The System Actions Panel

11. From the context menu (Figure 42–61), select **Human Task** and then the **Task Header** drop handler. The *Header* panel appears (Figure 42–64).

Figure 42–64 The Header Panel



42.4.3.1.2 Integrating the ADF BC Components First, create a database connection as follows:

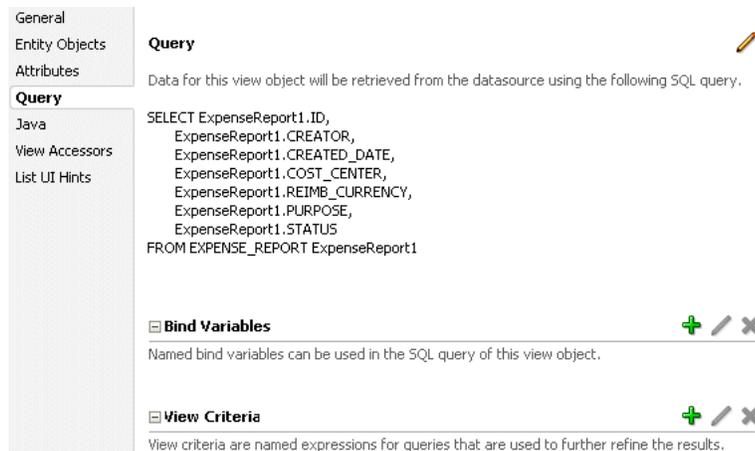
1. Right-click *ExpenseAppHumanTask* project in the Application Navigator and then select **New** from the context menu.
2. In the *New Gallery's Category* pane, expand **Business Tier** (if needed) and then select **ADF Business Components**.
3. Select **Business Components from Tables** in the *Items* pane. The *Initialize Business Components Project* page appears.
4. In the *Initialize Business Components Project* page, select **SQL92** from the *SQL Flavor* list.
5. Select **Java** from the *Type Map* list.
6. Click the **Add** icon (represented as a green plus sign in Figure 42–6) next to the *Application Connection* list. The *Create Database Connection* page appears.
7. Complete the *Database Connection* page as follows:
 - Select **Application Resources** (the default).
 - Enter *ExpenseTableConn* in the **Connection Name** field.
 - Select **Oracle (JDBC)** (the default) from the *Connection Type* list.
 - Enter the user name and password to the database. Use the "expense" user name and password. For more information, refer to "Creating a New User".
 - Select **thin** as the driver type.
 - Enter the host name
 - Enter *orcl* in the *Service Name* field.
8. Click **Test Connection**. If you defined the database connection properly, *Success!* appears. The *Initialize Business Components Project* page reappears.

- Click **OK**. The *Create Business Components from Tables* wizard appears and opens to the *Entity Objects* page (Step 1).

42.4.3.1.3 Creating the Business Components In the *Create Business Components from Tables* wizard:

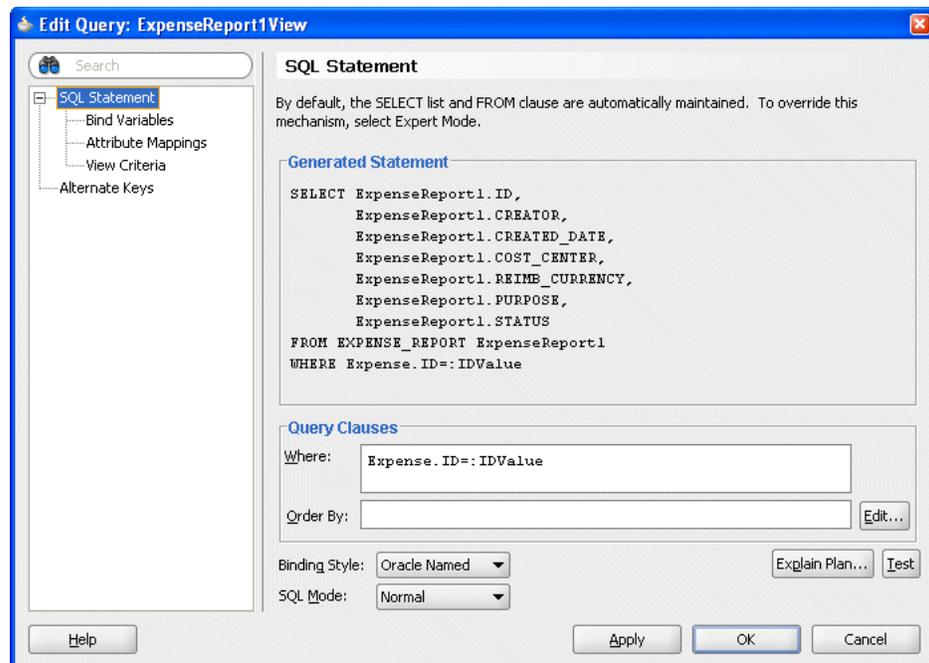
- In the *Entity Objects* page, enter *Expense%* in the *Name Filter* field.
- Click **Query**. The *EXPENSE_ITEMS* and *EXPENSE_REPORT* tables appear in the *Available* pane.
- Click **Next** to proceed to the Read-Only View Objects page. Complete the *Entity Objects* page as follows:
 - Enter *Expense%* in the *Name Filter* field.
 - Click **Query**. The *EXPENSE_ITEMS* and *EXPENSE_REPORT* tables appear in the *Available* pane.
 - Click **Add All** to move these tables to the *Selected* pane.
- Click **Finish** to complete the wizard. The business components created from the database tables display in the *Application Navigator*.
- In the *Application Navigator*, open *ExpenseReportView.xml* (located in *expenseapphumentaskflow*). *ExpenseReportView.xml* opens, defaulting to the *General* page.
- Select **Query**. The *Query* page opens (Figure 42–65).

Figure 42–65 The Query Page



- Click **Edit** (the pencil icon, illustrated in Figure 42–3). The *Edit Query* dialog appears.
- Append the following *WHERE* clause in the *Where* field (illustrated in Figure 42–66):
`Expense.ID = :IDValue`
- Click **OK**.

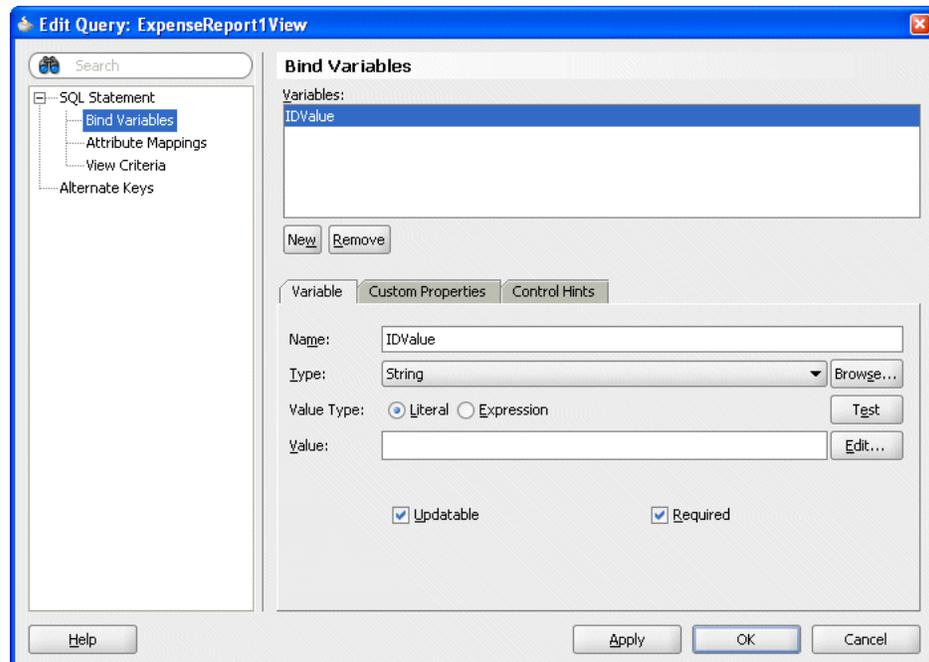
Figure 42–66 Appending the WHERE Clause to the SQL Statement



10. Click **Bind Variables** in the navigation tree. The *Bind Variables* page appears.

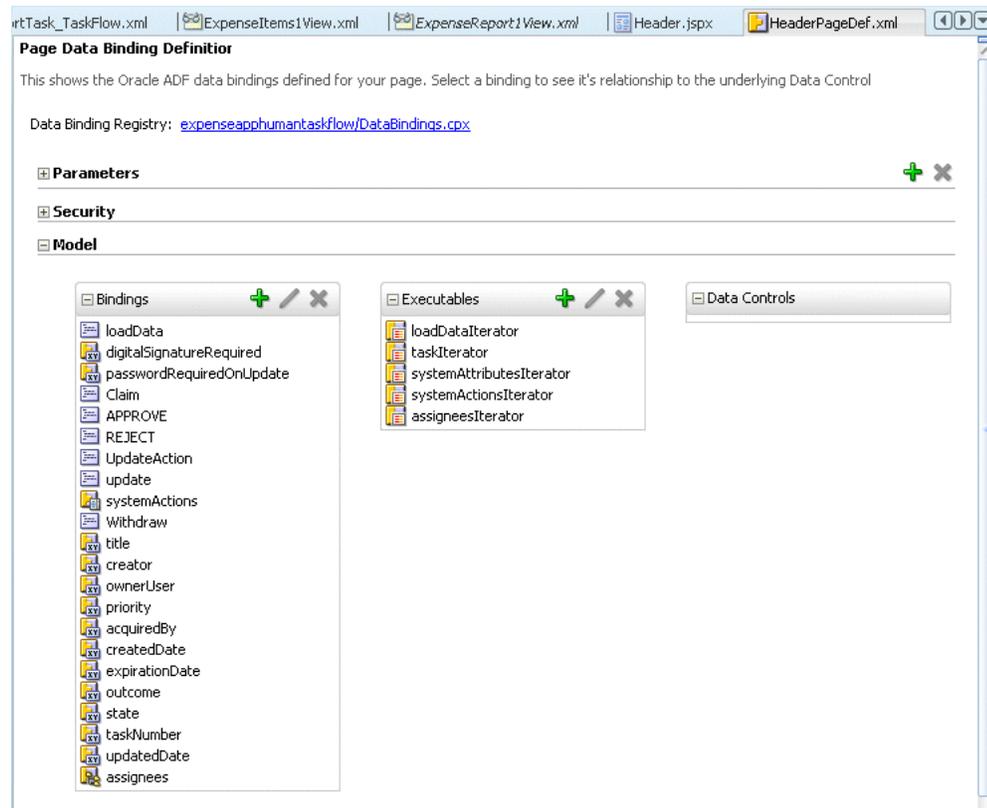
11. Click **New**.

Figure 42–67 Creating the Bind Variable

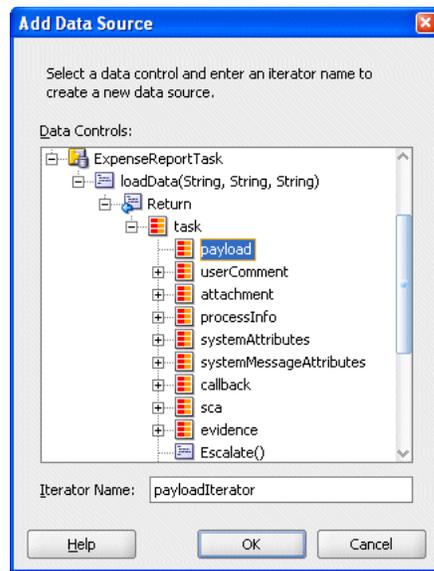


12. Enter *IDValue* in the *Name* field and then click **OK**.

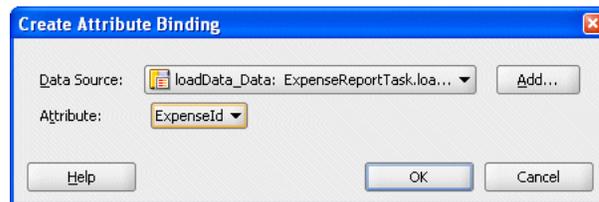
13. In *Header.jspx* (illustrated in Figure 42–64), invoke the context menu and then select **Go the Page Definition**. *HeaderPageDef.xml*, which contains the page configuration appears (Figure 42–68).

Figure 42–68 HeaderPageDef.xml

14. Click **Add** in the *Bindings* panel. The *Insert Item* dialog appears.
15. Select **General Bindings** as the category type.
16. Select **AttributeValues** as the item and then click **OK**. The *Create Attribute Binding* dialog appears.
17. Add a new data source for this binding by clicking **Add**. The *Add Data Source* dialog appears.
18. Expand *ExpenseReportTask* and then traverse to **payload** (*ExpenseReportTask* > *loadData* (*string, string, string*) > *Return* > *task* > *payload*). *payloadIterator* displays in the *Iterator Name* field.

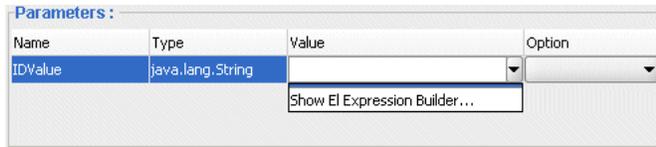
Figure 42–69 Selecting the Data Source for the Attribute Binding

19. Click **OK**. The *Create Attribute Binding* dialog reappears, with the payload data source displaying in the *Data Source* field (Figure 42–70).

Figure 42–70 The Create Attribute Binding Dialog

20. Select **ExpenseID** from the *Attribute* list and then click **OK**.
21. Click **Add** in the *Bindings* panel. The *Insert Item* dialog appears.
22. Select **General Bindings** as the category type.
23. Select **action** as the item and then click **OK**. The *Create Attribute Binding* dialog appears.
24. If needed, expand *AddModuleDataControl* and then select **ExpenseReportView**.
25. Click **New** to create a new iterator binding for *ExpenseReportView*. The *Iterator ID* dialog appears.
26. Click **OK**.
27. Select **ExecuteWithParams** from the *Operation* list.
28. In the *Parameters* section, click the *Value* field that corresponds to *IDValue* to invoke the list function (Figure 42–71).

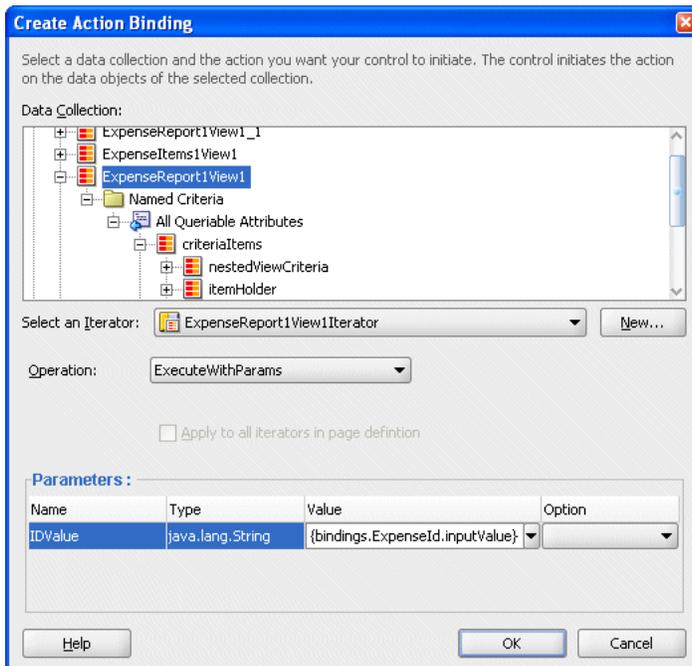
Figure 42–71 *Selecting the EL Expression Builder*



29. Select **Show EL Expression Builder...** The Expression Builder appears.
30. Create the following expression by first expanding *ADF Bindings* in the *Variables* pane and then traversing to **inputValue** (*ADF Bindings > bindings > ExpenseId > inputValue*).

```
{bindings.ExpenseId.inputValue}
```
31. Click **Insert Into Expression**.
32. Click **OK**. The expression appears as the value for the *IDValue* parameter (Figure 42–72).

Figure 42–72 *The Completed Create Action Binding Dialog*



33. Click **Add** in the *Execute* panel of *HeaderPageDef.xml* (Figure 42–68). The *Insert Item* dialog appears.
34. Select **invokeAction** as the category type and then click **OK**. The *Insert InvokeAction* dialog appears and defaults to the *Common Properties* page (Figure 42–73).

Figure 42–73 The Common Properties Page

The screenshot shows a dialog box titled "Insert invokeAction - Step 1 of 2". The main area is labeled "Common Properties". On the left, there is a tree view with "Common Properties" selected and "Advanced Properties" below it. The main area contains two fields: "id *" with the text "InvokeExecuteAction" and "binds *" with a dropdown menu showing "ExecuteWithParams". At the bottom, there are buttons for "Help", "< Back", "Next >", "Finish", and "Cancel".

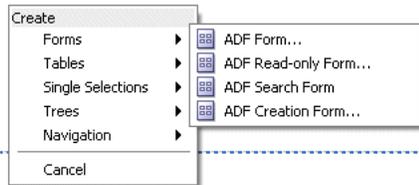
35. Complete the *Common Properties* by as follows:
 - Enter *InvokeExecuteAction* in the *Id* field.
 - Select **ExecuteWithParams** from the *Binds* list.
36. Proceed to the *Advanced Properties* page by clicking **Advanced Properties**.
37. Select **always** from the *Refresh* list and then click **Finish** (Figure 42–74).

Figure 42–74 The Advanced Properties Page

The screenshot shows a dialog box titled "Insert invokeAction - Step 2 of 2". The main area is labeled "Advanced Properties". On the left, there is a tree view with "Common Properties" and "Advanced Propert.." (Advanced Properties). The main area contains three fields: "Refresh:" with a dropdown menu showing "always", "RefreshAfter:" with an empty text box, and "RefreshCondition:" with an empty text box and a small menu icon. At the bottom, there are buttons for "Help", "< Back", "Next >", "Finish", and "Cancel".

38. From the data controls panel, drag and drop the *ExpenseReportView* node into *Header.jspx*.
39. From the context menu, first select **Forms** and then click **ADF Read-only Form**. The *Edit Form Fields* dialog displays.

Figure 42–75 The Context Menu

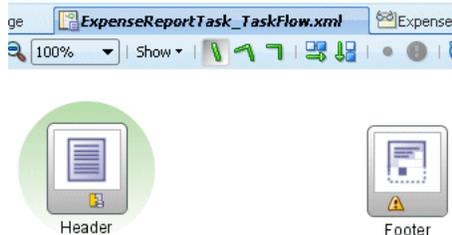


40. Click **OK**.

42.4.3.1.4 Creating the Footer Task Flow Object To create the Footer Task Flow object:

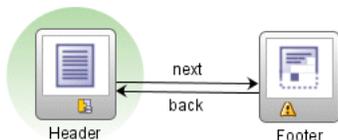
1. Drag and drop the *ExpenseReportView* node into *Header.jspx*.
2. From the context menu, first select **Tables** and then **ADF Read-only Table**. The *Edit Table Columns* dialog appears.
3. Click **OK**.
4. Create the footer for the page. Drag and drop a view object from the component panel into *In ExpenseReportTask_Taskflow.xml* (Figure 42–58).
5. Rename the view object *Footer* (Figure 42–76).

Figure 42–76 The Footer View Object

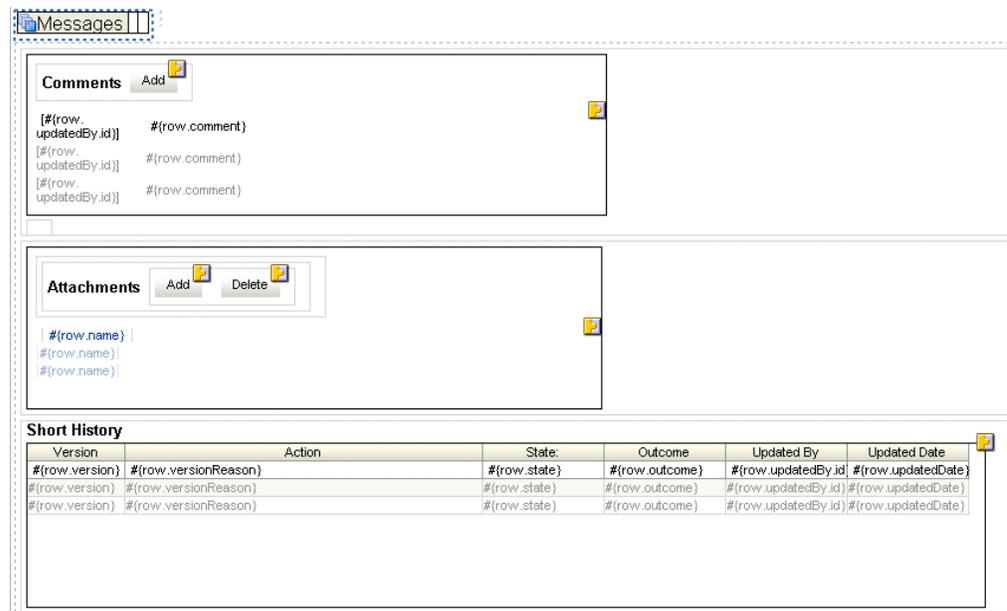


6. Connect the Header and Footer view objects as follows:
 - Drag and drop a *Control Flow Case* object from the *Components* panel into the *TaskFlow.xml*. Draw the flow from the Header object to the Footer object and then rename the flow as *next*.
 - Drag and drop a *Control Flow Case* object from the *Components* panel into the *TaskFlow.xml*. Draw the flow from the Footer object to the Header object and then rename the flow as *next* (Figure 42–77).

Figure 42–77 Connecting the Header and Footer Objects with Task Flows



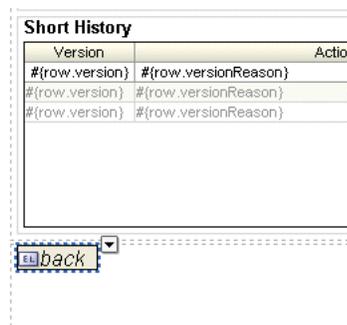
7. Double-click the *Footer* view object to create *Footer.jspx*.
8. Add *Custom* and *System* actions to *Footer.jspx* as described in "Populating the Header Task Flow Object".
9. Populate *Footer.jspx* with comments and history panels using the Human Task drop handlers as follows:
 - a. Drag and drop the task node (located under *ExpenseReportTask*, illustrated in Figure 42–60) into *Footer.jspx*. In the context menu that appears, select **Human Task**. The *Task Handler* context menu appears (Figure 42–61).
 - b. Select the **Task Comments** drop handler to add a comments section to *Footer.jspx*. The *Edit Action Bindings* dialog appears. Click **OK**.
 - c. Drag and drop the task node into *Footer.jspx*. In the context menu that appears, select **Human Task**. The *Task Handler* context menu appears (Figure 42–61).
 - d. Select the **Task Attachments** drop handler to add the attachments section to *Footer.jspx*.
 - e. Drag and drop the task node into *Footer.jspx*. From the context menu that appears, select **Human Task**. The *Task Handler* context menu appears.
 - f. Select **Task History** to add the history panel to *Footer.jspx*.

Figure 42–78 *Footer.jspx*

10. Add a *Back* button to *Footer.jspx* as follows:
 - a. Drag and drop a button component from the component panel. A *commandButton* component appears.
 - b. Double click the *commandButton* component to invoke the *Bind Action Property* dialog (Figure 42–79).
 - c. Select **Outcome Action**.
 - d. Select **back** and then click **OK**.

Figure 42–79 Bind Action Property for the Back Button

- e. Click the `commandButton` component and then rename it `back` (Figure 42–80).

Figure 42–80 The Back Button

11. Open `Header.jspx`.
12. Add a `Next` button as follows:
 - a. Drag and drop a button component from the component panel. A `commandButton` component appears.
 - b. Double-click the `commandButton` component to invoke the `Bind Action Property` dialog.
 - c. Select **Outcome Action**.
 - d. Select `back` and then click **OK**.
13. Click the `commandButton` component and then rename it `next`.
14. In the Application Navigator, expand Application Sources and then open `taskflow.properties`.
15. Add the following parameter:

```
human.task.lookup.type=LOCAL
```

42.4.3.1.5 Creating the Deployment Profile To create the deployment profile:

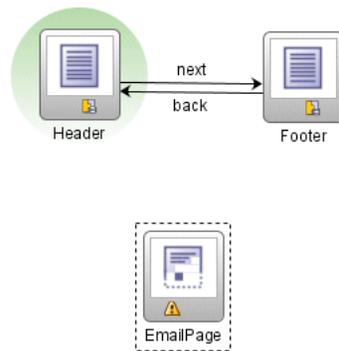
1. In the Application Navigator, select the `ExpenseAppHumanTaskFlow` project.
2. Left-click the project. The context menu appears.
3. Select **Project Properties**. The `Project Properties` dialog appears.
4. Select **Deployment** and then click **New**. The `Create Deployment Profile` dialog appears.
5. Complete the `Deployment Profile` dialog as follows:

- Select **WAR File** from the Archive Type list.
 - Enter *ExpenseAppHumanTaskFlowApp* in the *Name* field.
 - Click **OK**. *ExpenseAppHumanTaskFlowApp* appears in the Deployment Profiles pane of the Project Properties dialog.
6. Select *ExpenseAppHumanTaskFlowApp* and then click **Edit**. The WAR Deployment Profiles Properties dialog appears.
 7. Complete the WAR Deployment Profiles Properties dialog as follows:
 - a. Select **Specify Java EE Web Context Root**.
 - b. Enter *ExpenseAppHumanTaskFlowApp* in the corresponding field.
 - c. Click **OK**.

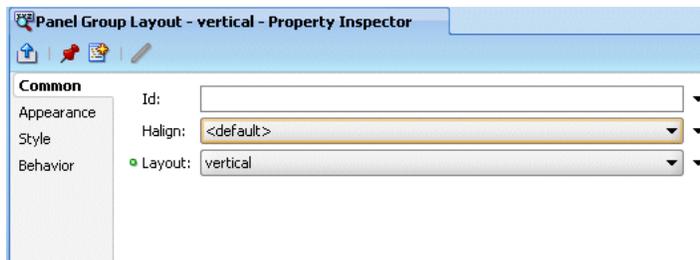
42.4.3.1.6 Designing the Notification Page To change the content contained in a workflow-generated notification:

1. Open *ExpenseReportTask_Taskflow.xml* (Figure 42-77)
2. Drag and drop a View object from the Components palette into *ExpenseReportTask_Taskflow.xml*.
3. Rename the object *EmailPage* (Figure 42-81).

Figure 42-81 Creating the EmailPage View Object



4. Double-click *EmailPage* view object to create *EmailPage.jspx*. The *Create JSF JSP* dialog appears.
5. Click **OK**.
6. Drag a *Panel Group Layout* component from the Component Palette and drop it into *EmailPage.jspx*.
7. Using the Property Inspector for the Panel Group Layout object, change the layout of the Panel Group Layout object by selecting **vertical** from the *Layout* list (Figure 42-82).

Figure 42–82 Changing the Layout

Tip: The Property Inspector's *Style* tab enables you to create user-defined styles.

8. Insert an image into *Email.jspx* before the Panel Group Layout object as follows:
 - a. Right-click the Panel Group Layout object and then select **Insert Before Panel Group Layout-vertical**.
 - b. From the context menu, select **ADF Faces**. The *Insert ADF Faces Item* dialog appears.
 - c. Select **Image** and then click **OK**. The *Insert Image* dialog appears and defaults to the *Common Properties* page.
 - d. Using the *Search* icon (Figure 42–54), locate the image and then click **Finish**.
9. Populate the Panel Group Layout object by dragging *ExpenseReportView1* (located beneath *AppModuleDataControl* in the Application Navigator's Data Controls) into *Email.jspx*.
10. Select **Forms** and then **ADF Read only Form** from the context menu (Figure 42–75).
11. Select the **Panel Form Layout**. The Panel Group Layout is populated.

Figure 42–83 Adding a Read Only Form

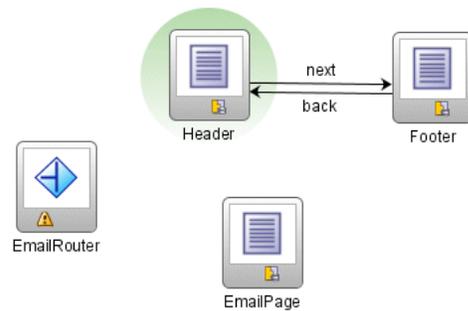
12. In *Panel Group Layout -vertical- Property Inspector* (Figure 42–84), select **Style** and then define the inline style as follows:


```
"margin: 2px; border: 1 px solid black; padding: 3px; overflow: auto; width: 95%;"
```

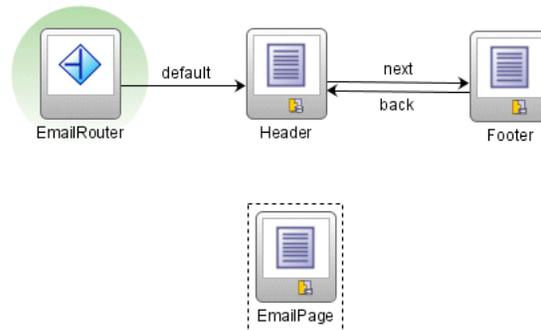
Note: Separate each style with semi-colons. Separate each key value with quotation marks.

Figure 42–84 Defining the Inline Style

13. Drag a Router object from the Components Palette into *ExpenseReportTask_TaskFlow.xml*.
14. Using the Property Inspector for the router object, rename the router object by entering *EmailRouter* in the *id** field.

Figure 42–85 Adding a Router

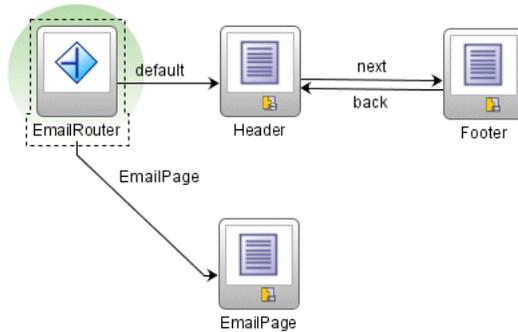
15. In the Property Inspector for Email Router (*router-EmailRouter-Property Inspector*), enter *default* in the *default-outcome** field.
16. Drag a *Control Flow Case* component from the Components Palette and then connect it to the *EmailRouter* component.
17. Connect the *Control Flow Case* component to the *Header* object (Figure 42–86).

Figure 42–86 Connecting EmailRouter to Header

18. In the Property Inspector for the *Call Flow Case* component (*control-flow-case-Header-Property Inspector*), select **default** from the *from outcome* list. The *from-outcome* and *to-activity-id* values are *default* and *Header*, respectively.
19. Drag a *Control Flow Case* component from the Components Palette and then connect it to the *EmailRouter* component.

20. Rename this component *EmailPage* and then connect this Control Flow Case component using the Property Inspector as by entering *EmailPage* as both the *from-outcome* and *to-activity-id* values.

Figure 42–87 Attaching EmailRouter to EmailPage



21. Select the EmailRouter.
22. In the *Common* tab of the EmailRouter's Property Inspector (*router-EmailRouter-Property Inspector*), expand the *Cases* section (if needed) click **Add**. *outcome1* appears. in the *outcome** field.
23. Select **EmailPage** from the *outcome* list.
24. Select **Expression Builder** from the list accessed from the *expression* field. The Expression Builder appears.
25. Build the following expression and then click **OK**.

```
# {pageFlowScope.bpmClientType=="notificationClient" }
```

42.4.3.1.7 Deploying the TaskFlow Application See [Section 42.2.2, "Deploying the ADF BC Application \(ExpenseReportADFBCApp\)"](#).

Composite Test Framework

This part describes how to create, deploy, and run test cases that automate the testing of SOA composite applications.

This part contains the following chapter:

- [Chapter 43, "Testing SOA Composite Applications"](#)

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 prior to 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 contains the following topics:

- [Section 43.1, "Overview of the Composite Test Framework"](#)
- [Section 43.2, "Components of a Test Suite"](#)
- [Section 43.3, "Creating Test Suites and Test Cases in Oracle JDeveloper"](#)
- [Section 43.4, "Creating the Contents of Test Cases"](#)
- [Section 43.5, "Deploying a Test Suite"](#)
- [Section 43.6, "Running a Test Suite and Viewing Report Results"](#)

See Also: Composite test sample files located in the `soa-samples.zip` file under `samples\test\CompositeTestApp`. Open the `CompositeTestApp.jws` file in Oracle JDeveloper to automatically load these samples.

43.1 Overview of 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
- Calculates the percentage of BPEL process service component source code executed in terms of the percentage of simple activities executed
- Creates reports of test results

The following sections provide an overview of test suite concepts:

- [Section 43.1.1, "Test Cases Overview"](#)
- [Section 43.1.2, "Test Suites Overview"](#)
- [Section 43.1.3, "Emulations Overview"](#)

- [Section 43.1.4, "Assertions Overview"](#)
- [Section 43.1.5, "BPEL Process Code Coverage Overview"](#)

43.1.1 Test Cases Overview

The test framework supports two types of unit tests:

- **Component test** — Consists of testing at the individual BPEL process service component level. For Applications Drop 4 and Beta 3, service component level testing is *not* supported.
- **Composite test** — Consists of testing at the higher SOA composite application level. In this type of testing, wires, binding component services, service components (such as BPEL processes and mediator), and binding component references are tested.

See Also: [Section 43.3, "Creating Test Suites and Test Cases in Oracle JDeveloper"](#) on page 43-6

43.1.2 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.

See Also:

- [Section 43.3, "Creating Test Suites and Test Cases in Oracle JDeveloper"](#) on page 43-6
- [Section 43.4, "Creating the Contents of Test Cases"](#) on page 43-8

43.1.3 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.

See Also:

- [Section 43.2.2, "Emulations"](#) on page 43-4
- [Section 43.4, "Creating the Contents of Test Cases"](#) on page 43-8

43.1.4 Assertions Overview

Assertions enable you to verify variable data or process flow. You can perform the following types of assertions:

- **Simple Value Assert** — Compare the value of a selected string or number variable to an expected value. For example, compare a customer name or loan amount in a loan request to a specific value.

- XML Assert — 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.

See Also:

- [Section 43.2.3, "Assertions"](#) on page 43-5

43.1.5 BPEL Process Code Coverage Overview

Code coverage provides a method for calculating the completeness of the executed tests. This is calculated as the percentage of simple activities executed at least once, compared to the number of simple activities defined in a BPEL process service component in a SOA composite application. Simple activities are nonstructured activities such as invoke, receive, reply, and assign activities.

Note: Code coverage is *only* calculated on the activities of BPEL process service components. Code coverage on other service component types is not supported.

See Also: [Section 43.6, "Running a Test Suite and Viewing Report Results"](#) on page 43-22

43.2 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.

This section contains the following topics:

- [Section 43.2.1, "Process Initiation"](#)
- [Section 43.2.2, "Emulations"](#)
- [Section 43.2.3, "Assertions"](#)
- [Section 43.2.4, "Message Files"](#)

43.2.1 Process Initiation

You first define the operation of your process in a binding component service such as a SOAP web service. The following example defines the operation of `initiate` to initiate the `TestFwk` SOA composite application. The initiation payload is also defined in this section:

```
<?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>qing.zhong@oracle.com</email>
            <customerName>Joe Smith</customerName>
          </loanApplication>
        </content>
      </part>
    </message>
  </initiate>
</compositeTest>
```

```

        <loanAmount>2000</loanAmount>
        <carModel>Camry</carModel>
        <carYear>2007</carYear>
        <creditRating>800</creditRating>
    </loanApplication>
</content>
</part>
</message>
</initiate>
</compositeTest>

```

43.2.2 Emulations

You create emulations to simulate the message data that your SOA composite application receives from Web service partners.

The following test code instructs Oracle SOA Suite to initiate the loan request with an error and receive a fault message in return from a Web service partner:

```

<?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 a fault message:

```

<error xmlns="http://services.otn.com">
  Invalid SSN, SSN can not start with digit '0'.
</error>

```

See Also:

- [Section 43.4, "Creating the Contents of Test Cases"](#) on page 43-8

43.2.3 Assertions

You create assertions to validate a variable or an entire XML document at a point during SOA composite application execution. The following example instructs Oracle SOA Suite to ensure that the content of the `customerName` variable matches the content specified.

```
<?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>
```

See Also:

- [Section 43.4, "Creating the Contents of Test Cases"](#) on page 43-8

43.2.4 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 user interface. 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>
```

See Also:

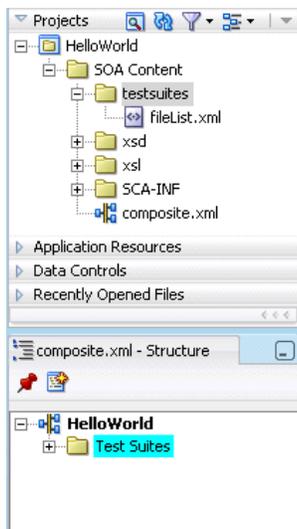
- [Section 43.4, "Creating the Contents of Test Cases"](#) on page 43-8 for instructions on loading message files into test mode

43.3 Creating Test Suites and Test Cases in Oracle JDeveloper

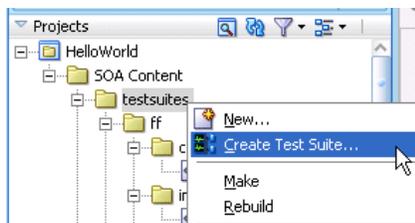
This section describes how to create test suites and their test cases for a SOA composite application in Oracle JDeveloper. The test cases consist of sets of commands to perform as the test instance is executed.

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.

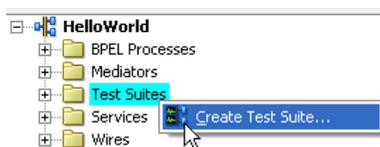
1. Open the SOA Composite Editor.
2. Open the SOA composite application in which to create a test suite.
3. Go to the Application Navigator or **Structure** window. If the **Structure** window does not appear, select **Structure** from the **View** main menu.



4. Create a test suite in either of two ways:
 - a. Right-click **testsuites** in the **Application Navigator** and select **Create Test Suite**.



- b. Right-click **Test Suites** in the **Structure** window and select **Create Test Suite**.



5. Enter a test suite name (for example, `logicTest`).
6. Click **OK**.

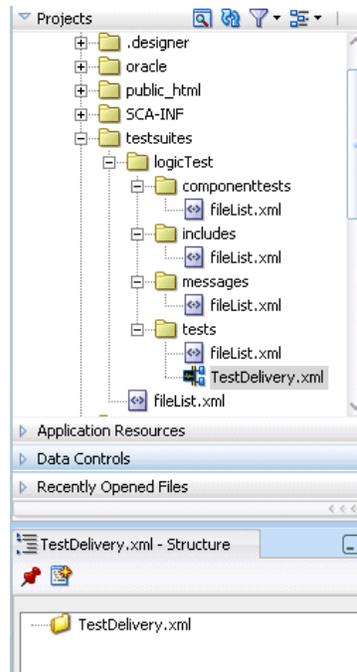
The Create Composite Test window appears.

7. Enter a test name (for this example, **TestDelivery** is entered) and an optional description, and click **OK**.

This action creates a test named **TestDelivery.xml** in the **Applications Navigator**, along with the following subfolders:

- **componenttests** — contains BPEL process service component test files (*not* supported for beta 3)
- **includes** — contains `include` files that you create
- **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. 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 windows of service components or dropping service components into the editor, can be performed in test mode.



The following operating system test suite directory is also created:

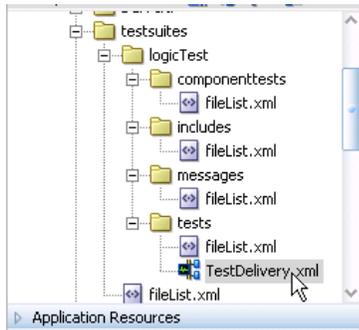
```
C:\JDeveloper\mywork\application_name\project_name\testsuites\logicTest
```

The following subdirectories for adding test files are created beneath `logicTest`: `includes`, `messages`, `componenttests`, and `tests`.

8. If you want to exit test mode and return to design mode in the SOA Composite Editor, click the **x** icon next to **TestDelivery.xml** above the designer.



9. Save your changes when prompted.
10. Return to test mode by double-clicking **TestDelivery.xml** under the **testsuites** folder in the **Application Navigator**.



Notes:

- The **componenttests** folder for the BPEL process service component is not supported for beta 3. Only testing at the SOA composite application level is supported for beta 3.
 - Do not edit the **FileList.xml** files that display under 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.
-
-

43.4 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 either message data, fault data, or both types that your SOA composite application receives from Web service partners. You create assertions to validate test data in a variable or XML document as a process is executed.

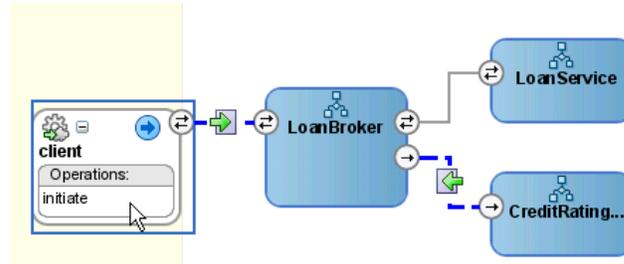
This section contains the following topics:

- [Section 43.4.1, "Initiating Inbound Messages"](#)
- [Section 43.4.2, "Emulating Outbound Messages"](#)
- [Section 43.4.3, "Emulating Callback Messages"](#)
- [Section 43.4.4, "Emulating Fault Messages"](#)
- [Section 43.4.5, "Creating Value or XML Assertions"](#)

43.4.1 Initiating 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.
2. Double-click the binding component service (for example, named **initiate**).

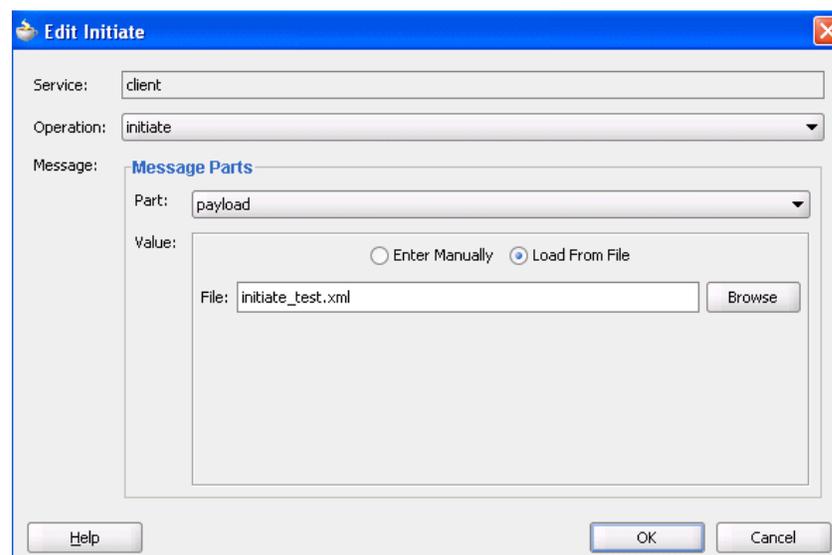


The Edit Initiate window appears.

3. Enter the following details:

Field	Value
Service	Displays the name of the binding component service (client).
Operation	Displays the operation type of the binding component service (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.

An example of this window is shown below:



An example of an inbound process initiation message from a client is as follows:

```
<?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:

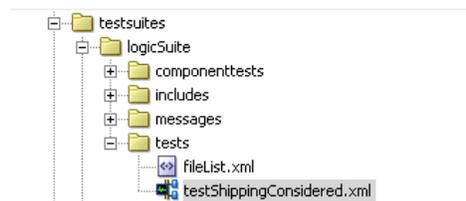
```
<loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
  <SSN>111222333</SSN>
  <email>qing.zhong@oracle.com</email>
  <customerName>Qing Zhong</customerName>
  <loanAmount>20000</loanAmount>
  <carModel>Camry</carModel>
  <carYear>2007</carYear>
  <creditRating>800</creditRating>
</loanApplication>
```

4. Click **OK**.

43.4.2 Emulating Outbound Messages

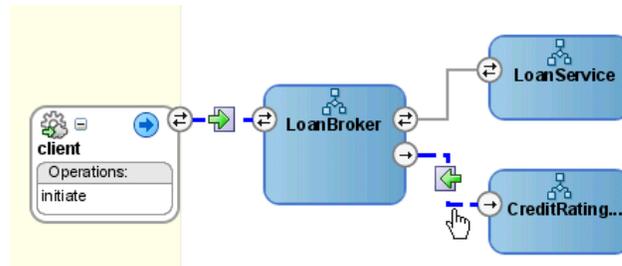
You can simulate a message returned from a synchronous Web service partner.

1. Go to the SOA Composite application in test mode.
2. Double-click a test case beneath the **Test Suites** folder in the **Application Navigator**.

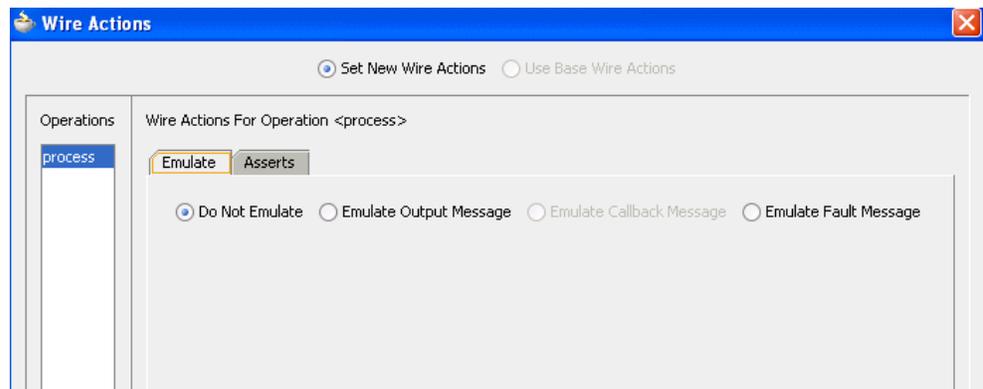


The SOA composite application in the SOA Composite Editor is refreshed to display in test mode. This mode enables you to define test information.

3. Double-click the wire of the SOA composite application area to test. For this example, the wire between the **LoanBroker** process and the synchronous **CreditRating** web service.



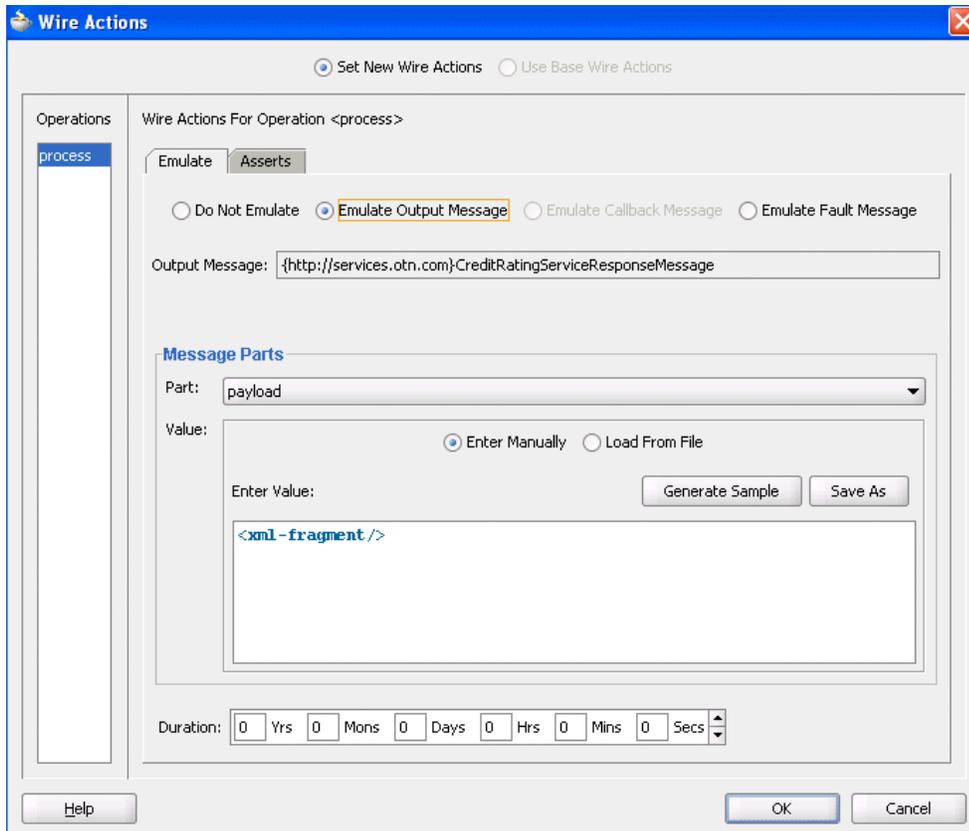
This displays the Wire Actions window, from which you can design emulations and assertions for the selected part of the SOA composite application. By default, **Do Not Emulate** is selected. This is because no tests have been created.



4. Click **Emulate Output Message**.
5. Enter the following details:

Field	Value
Output Message	Displays the output message of the selected service component.
Part	Select the message part containing the output (for example, payload).
Value	Create a simulated output 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 message to be delivered from the Web service partner.

An example of this window is shown below:



An example of a simulated output message from a synchronous Web service partner that you enter manually or load from a file is as follows:

```
<?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 callbackOperation="process" 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>
```

6. Click OK.

Notes:

- The **Use Base Wire Actions** button on the Wire Actions window is disabled for beta 3.
- The Create Component Test window that displays when you double-click a BPEL process service component in test mode (for example, **LoanBroker** or **CreditRatingService** shown in the examples in this chapter) is not supported for beta 3.

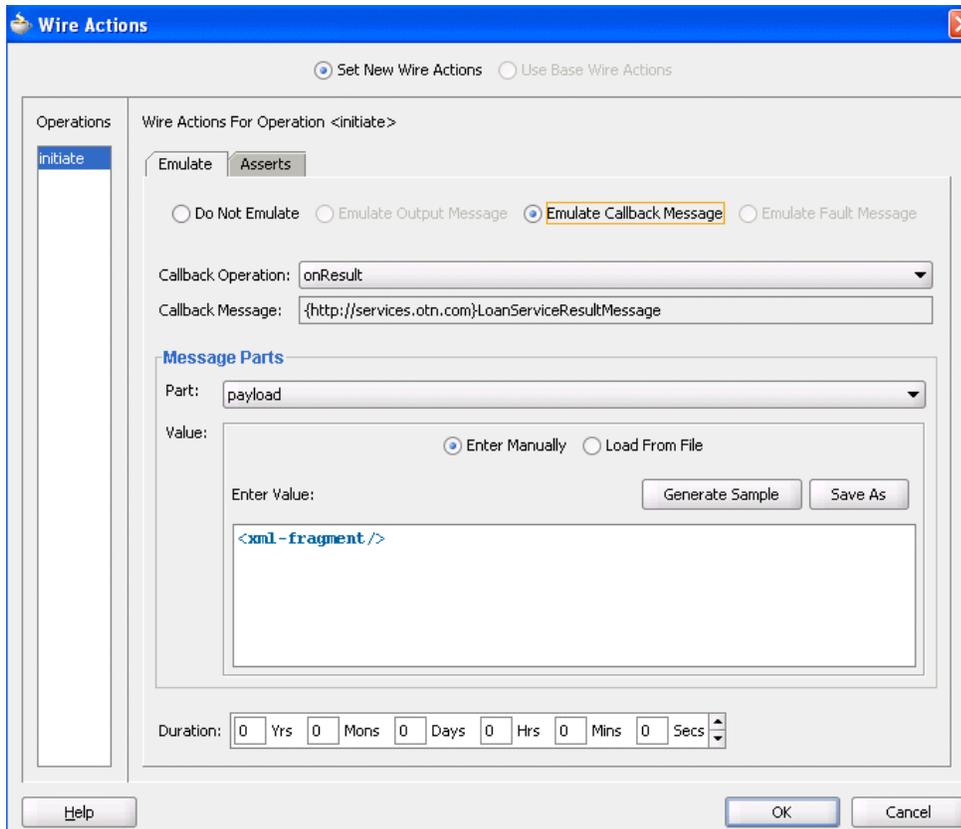
43.4.3 Emulating Callback Messages

You can simulate a callback message returned from an asynchronous Web service partner.

1. Access the Wire Actions window by following Step 1 through Step 3 on page 43-10.
2. Click **Emulate Callback Message**. This field is only enabled for asynchronous processes.
3. Enter the following details:

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.
Duration	Enter the maximum amount of time to wait for the callback message to be delivered from the Web service partner.

An example of this window is shown below:



An example of a simulated callback message from a Web service partner that you enter manually or load from a file is as follows:

```
<?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.

```
<loanOffer xmlns="http://www.autoloan.com/ns/autoloan">
  <providerName>Bank Of America</providerName>
```

```

    <selected>>false</selected>
    <approved>>true</approved>
    <APR>1.9</APR>
  </loanOffer>

```

4. Click **OK**.

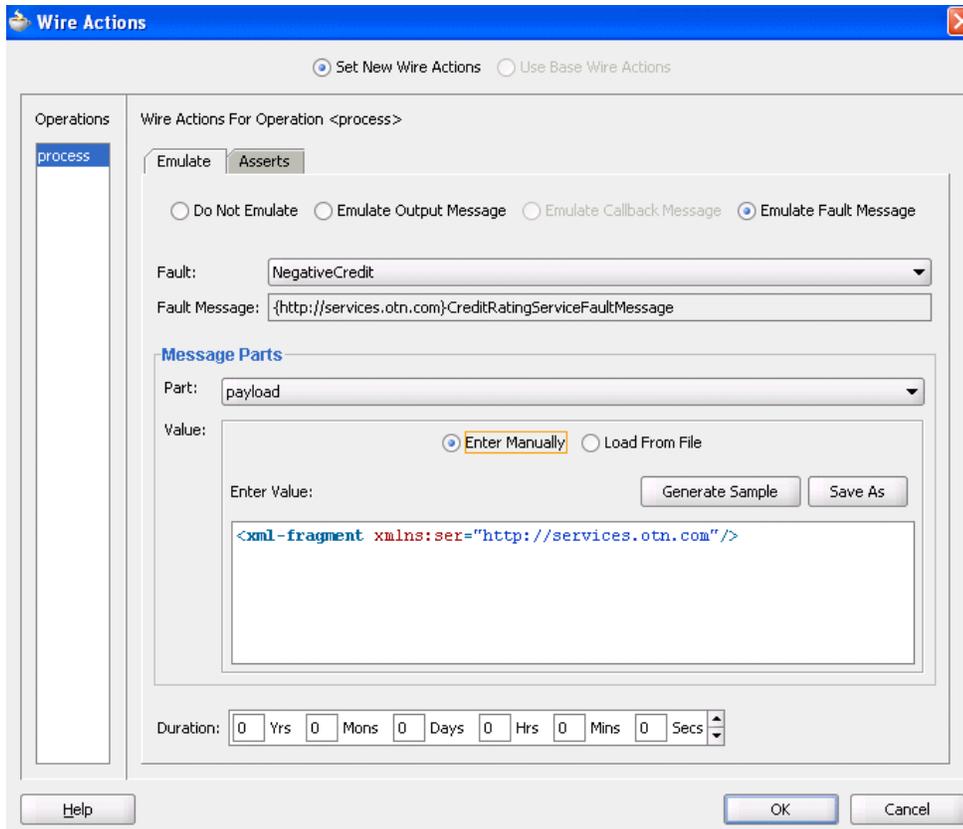
43.4.4 Emulating Fault Messages

You can simulate a fault message returned from a Web service partner. This enables you to test fault handling capabilities in your process.

1. Access the Wire Actions window by following Step 1 through Step 3 on page 43-10.
2. Click **Emulate Fault Message**.
3. Enter the following details:

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.

An example of this window is shown below:



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 43.2.2, "Emulations"](#) on page 43-4.

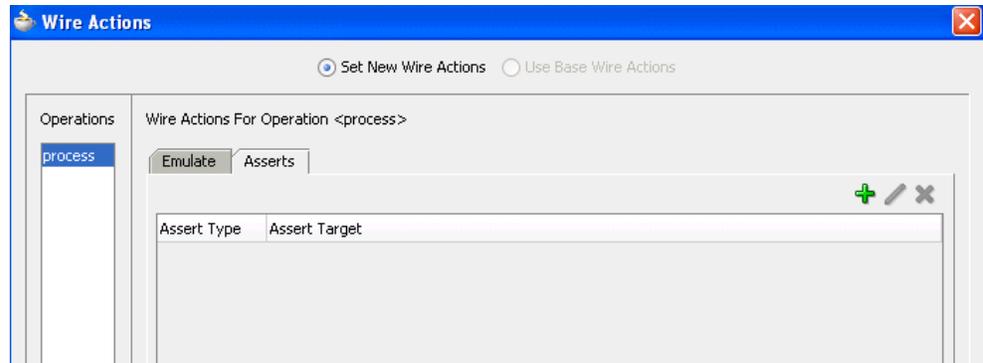
Note also that the operation for the SOA composite application (for this example, process) displays in the **Operations** column on the left.

4. Click **OK**.

43.4.5 Creating Value or XML Assertions

You perform assertions to verify variable data or process flow. Assertions enable you to validate test data in a variable or an entire XML document as a process is executed. This is done by extracting a value from a variable or an XML document and comparing it to an expected value.

1. Access the Wire Actions window by following Step 1 through Step 3 on page 43-10.
2. Click the **Asserts** tab.



3. Click the **Create** icon.

The Create Assert window appears.

4. Select the type of assertion to perform at the top of the window. 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.

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 Fault	Select to assert a fault into the application flow.

5. See the following section based on the type of assertion you want to perform.

For...	See Section...
Variable assertions	Section 43.4.5.1, "Variable Assertions" on page 43-17
XML assertions	Section 43.4.5.2, "XML Assertions" on page 43-19

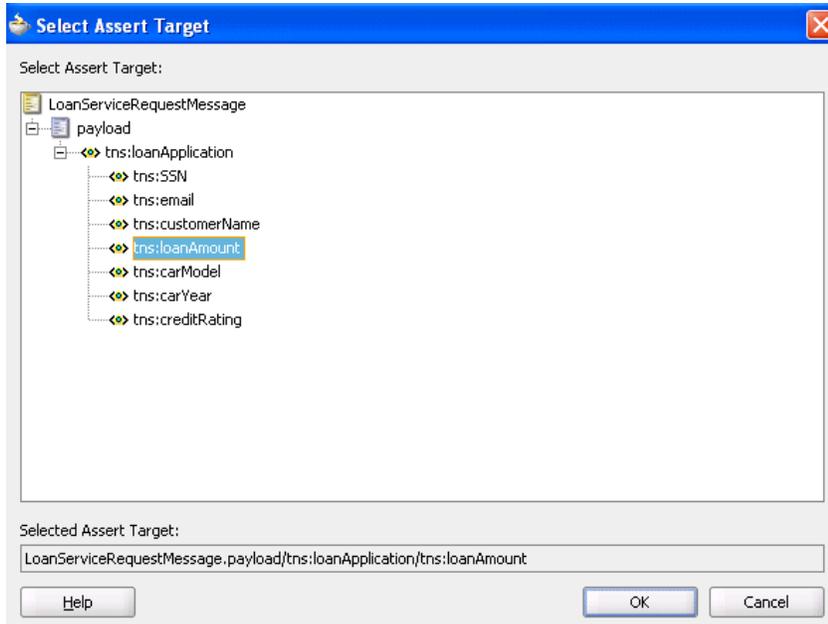
43.4.5.1 Variable Assertions

This test compares the value of a selected string or number variable to an expected value.

1. Click **Browse** to select the target variable to assert.

The Select Assert Target window appears.

2. Select a specific variable, and click **OK**. For example, select a variable such as **loanAmount** to perform a variable assertion.

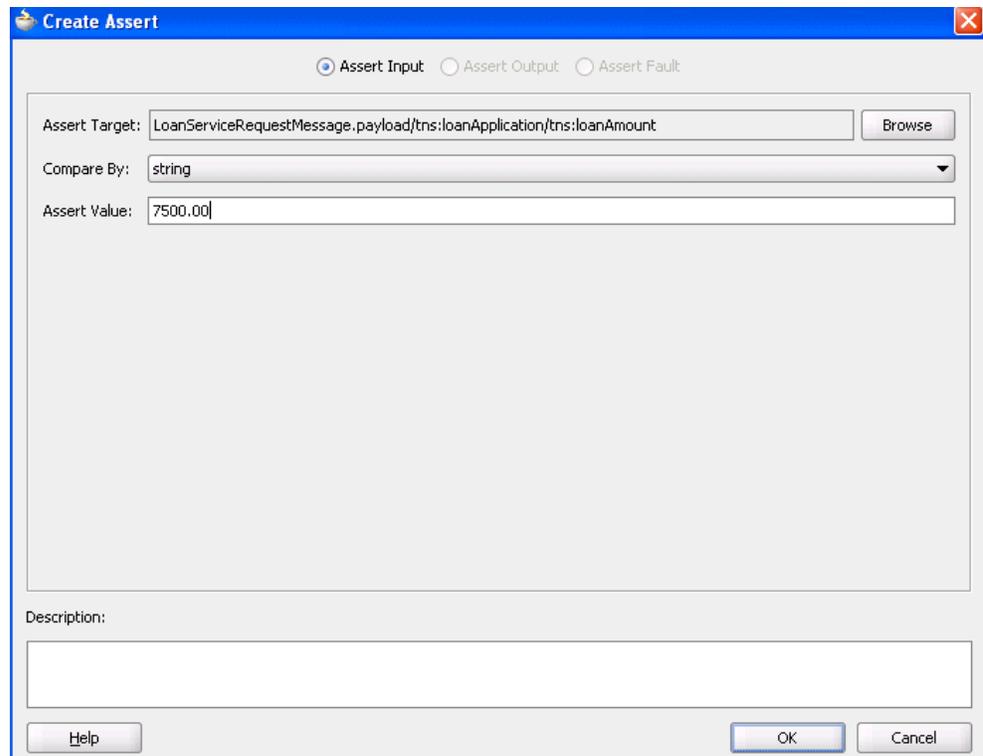


The Create Assert window refreshes based on your selection of a variable.

3. Enter the following details in the remaining fields:

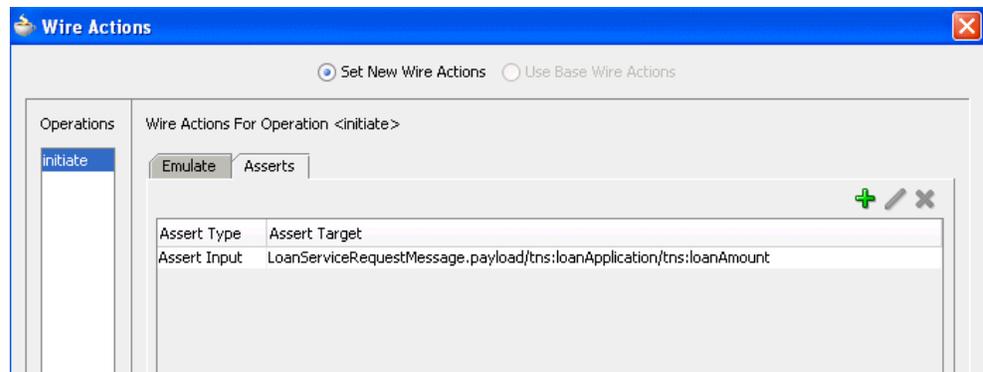
Field	Value
Fault	Select the type of fault to assert (for example, NegativeCredit). This field only displays if you select Assert Fault in Step 4 on page 43-17.
Assert Target	Displays the variable assert target you selected in Step 2.
Compare By	Select the type of comparison: <ul style="list-style-type: none"> ■ string — Compares string values ■ number — Compares numerical values ■ pattern-match — Compares a regular expression pattern (for example, [0-9]*).
Assert Value	Enter the value you are expecting. This value is compared to the value for the assert target.
Description	Enter an optional description.

An example of this window with **Assert Input** selected is shown below:



4. Click **OK**.

The Wire Actions window displays your selection.



5. Click **OK**.

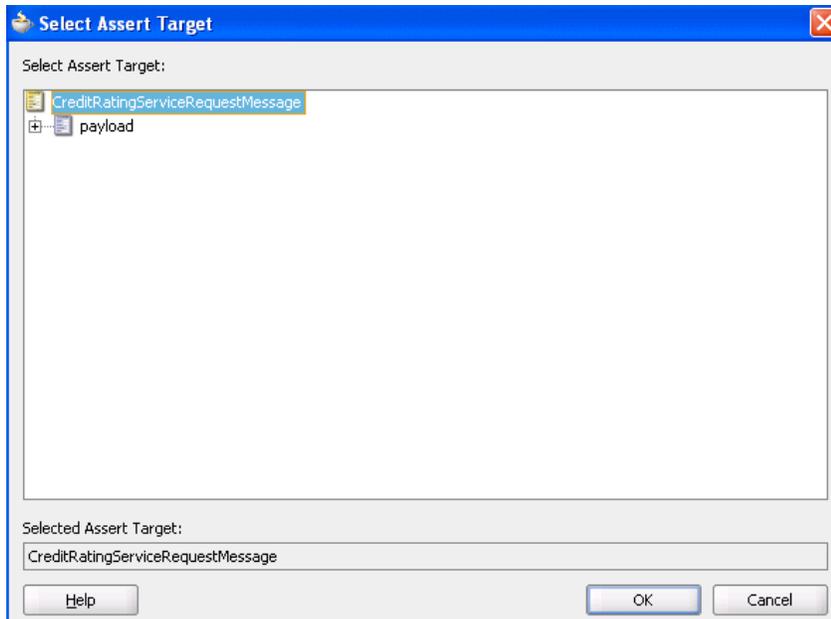
43.4.5.2 XML Assertions

This test compares the values of an entire XML document to the expected values.

1. Click **Browse** to select the target XML document to assert.

The Select Assert Target window appears.

2. Select the entire document, and click **OK**. For example, select an entire XML document (**CreditRatingServiceRequestMessage**) to perform an XML assertion.

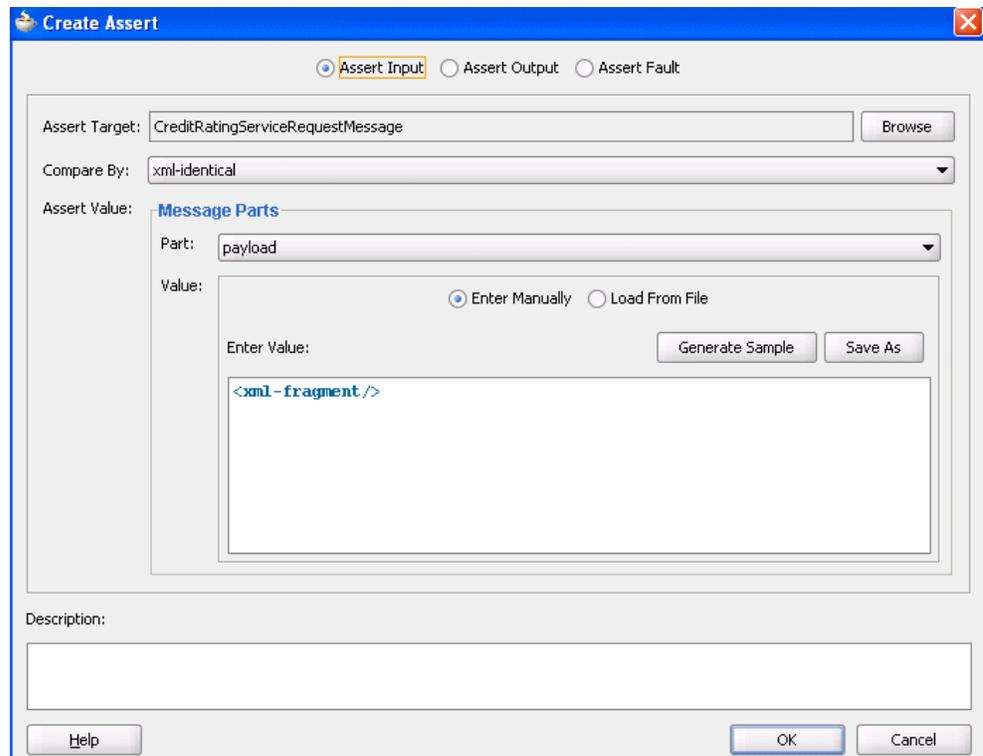


The Create Assert window refreshes based on your selection of an entire XML document.

3. Enter the following details in the remaining fields:

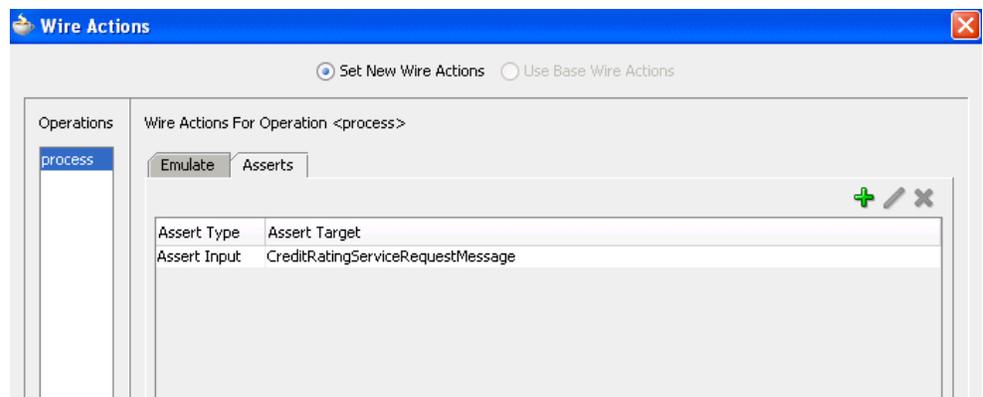
Field	Value
Fault	Select the type of fault to assert (for example, NegativeCredit). This field only displays if you select Assert Fault in Step 4 on page 43-17.
Assert Target	Displays the XML document assert target you selected in Step 2.
Compare By	Specify the strictness of the comparison. <ul style="list-style-type: none"> ▪ xml-identical — Used when the comparison must be exact (for example, element ordering). ▪ xml-similar — Used when the comparison must be similar in content, but do not need to exactly match
Part	Select the message part containing the XML document (for example, payload).
Value	Create an XML document whose content is compared to the assert target content: <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.

An example of this window with **Assert Input** selected is shown below:



4. Click OK.

The Wire Actions window displays your selection.



5. Click OK.

43.5 Deploying a Test Suite

After creating a test suite of test cases, you deploy the test to Oracle Enterprise Manager Fusion Middleware Control. Two deployment methods are provided:

- [Section 43.5.1, "Deploying from Oracle JDeveloper"](#)

43.5.1 Deploying from Oracle JDeveloper

Test suites are deployed as part of the SOA composite application for which you create a deployment profile in Oracle JDeveloper.

See Also: [Section 3.4.1, "Deploying Applications with Oracle JDeveloper"](#) on page 3-2 for instructions on creating and deploying a deployment profile in Oracle JDeveloper

43.6 Running a Test Suite and Viewing Report Results

After deployment, you can run the test cases of a test suite on a SOA composite application instance and view XML document reports. Two methods are provided:

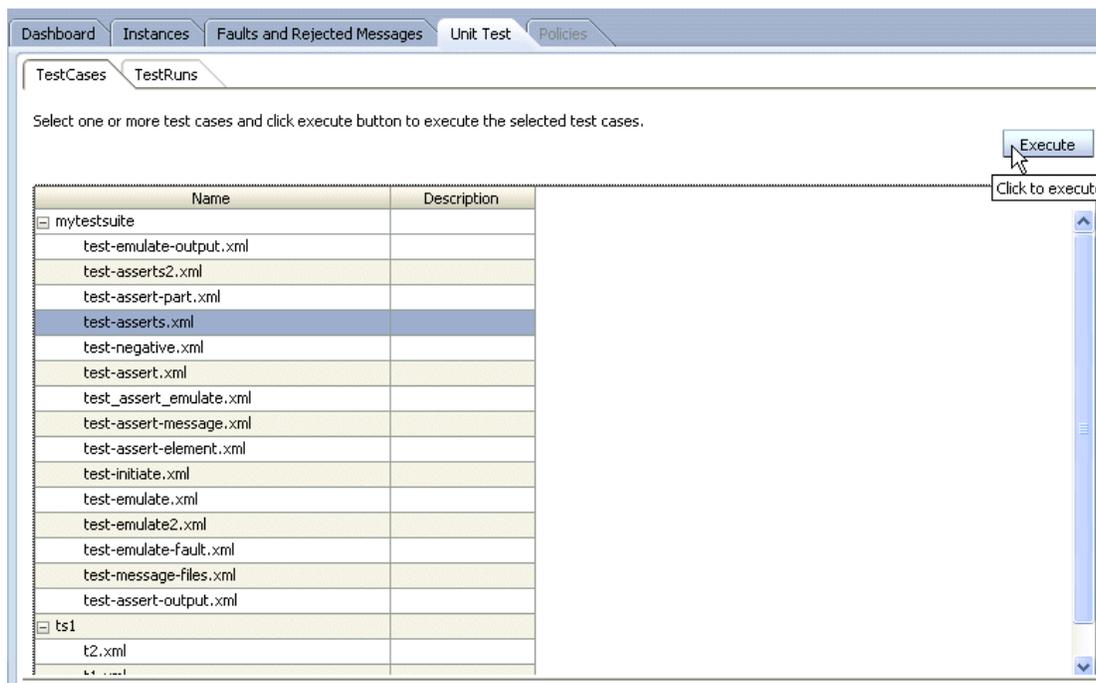
43.6.1 Running from Oracle Enterprise Manager Fusion Middleware Control

Oracle Enterprise Manager Fusion Middleware Control can be used to run test suites and generate report results.

1. Go to the Oracle SOA Console
`http://hostname:8888/SOAConsole`
2. Log in as `fmwadmin/password` if prompted.
where `password` is the `fmwadmin` password.
3. Create and run a test instance of the SOA composite application that includes the test suite. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for instructions on creating and running a test instance.
4. Click the **Unit Test** tab.

The page refreshes to display the test suites and test cases available with this instance.

5. Select entire test suites or individual test cases of test suites to run, and click **Execute**.



The Details of testrun window appears.

6. Enter the following details, and click **OK**.

Field	Description
TestRun Name	Enter a name for the test instance. When testing is complete, report details are captured under this name.
Timeout	Enter a value in seconds in which to complete this test. If the test does not complete within this time limit, then testing is terminated.
Number of Concurrent Test Instances	Enter the number of test instances to create.

The **Test Runs** tab displays details about the test instance.

- Click a specific test in the **TestRun Name** column.

Select a TestRun to view its details below

TestRun Name	ID	Start Time	End Time	Status	Success
negative-assert	e33880a97f8fcd6d:f26a8ad:113bb12b036:-7fdd	Thu Jul 12 09:10:07 PDT 2007	Thu Jul 12 09:10:07 PDT 2007	Failed	0
emulate-fault	e33880a97f8fcd6d:f26a8ad:113bb12b036:-7fdd	Thu Jul 12 09:17:11 PDT 2007	Thu Jul 12 09:17:11 PDT 2007	passed	10
assert-message	e33880a97f8fcd6d:f26a8ad:113bb12b036:-7fdd	Thu Jul 12 09:15:56 PDT 2007	Thu Jul 12 09:15:57 PDT 2007	passed	10
emulate-callback	e33880a97f8fcd6d:f26a8ad:113bb12b036:-7fdd	Thu Jul 12 09:13:04 PDT 2007	Thu Jul 12 09:13:04 PDT 2007	passed	10
asserts	e33880a97f8fcd6d:f26a8ad:113bb12b036:-7fdd	Thu Jul 12 08:48:00 PDT 2007	Thu Jul 12 08:48:01 PDT 2007	passed	10
assert	e33880a97f8fcd6d:f26a8ad:113bb12b036:-7fdd	Thu Jul 12 08:45:38 PDT 2007	Thu Jul 12 08:45:39 PDT 2007	passed	10
initiate	e33880a97f8fcd6d:f26a8ad:113bb12b036:-7fdd	Thu Jul 12 08:43:12 PDT 2007	Thu Jul 12 08:43:16 PDT 2007	passed	10

- Scroll down the page and review the results under the **Results** tab.

Results Coverage

Expand Test Suites to view status of each Test Case. Select a Test Suite or Test Case to see assertion details in the next section below.

Test Suites and Test Cases	Status
mytestsuite	
asserts.xml	passed

Assertion Details for mytestsuite

Display Failures Only

Composite Instance	Location	Type	Status	Expected Value	Actual Value	Error Message
3	client	Wire	passed	111222333	111222333	
3	client	Wire	passed	Qing Zhong	Qing Zhong	

- Click a composite instance number to view specific test details.
- Click the **Coverage** tab to view details about the percentage of source code executed on activities in the BPEL process service component.

TestCases TestRuns

Details of TestRun : asserts (ID : e33880a97f8fcd6d:f26a8ad:113bb12b036:-7fdd)

Total 1 Running 0 Completed 1 Passed 1 Failed 0 Errored 0 Success Rate 100% Refresh Test Status

Results Coverage

Source	Type	Coverage
LoanService	BPEL Component	100%
LoanBroker	BPEL Component	100%
CreditRatingService	BPEL Component	60%

- Click the percentage value under **Coverage** for a BPEL process service component.
Activities that were executed are framed in green and activities that were not executed are framed in pink.

Component  CreditRatingService  [3 Covered Activities](#)
 [2 Uncovered Activities](#)

Coverage Threshold

```

<!-- CreditRatingService BPEL Process -->
- <process name="CreditRatingService" targetNamespace="http://services.otn.com"
suppressJoinFailure="yes" xmlns:tns="http://services.otn.com"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
  <!-- List of services participating in this BPEL process -->
  - <partnerLinks>
    <!-- The 'client' role represents the requester of this service. -->
    <partnerLink name="client" partnerLinkType="tns:CreditRatingService"
myRole="CreditRatingServiceProvider"/>
  </partnerLinks>
  <!-- List of messages and XML documents used as part of this BPEL process -->
  - <variables>
    <!-- Reference to the message passed as input during initiation -->
    <variable name="input" messageType="tns:CreditRatingServiceRequestMessage"/>
    <!--
      Reference to the message that will be sent back to the
      requestor during callback
    -->
  
```

- Scroll down and note the pink sections that were not executed.

```

<receive name="receiveInput" partnerLink="client" portType="tns:CreditRatingService"
operation="process" variable="input" createInstance="yes"/>
<!-- switch depends on the input value -->
- <switch>
- <case condition="starts-with(bpws:getVariableData('input', 'payload','/tns:ssn'),'0') =
'true'">
- <sequence>
- <!--
Generate content of output message based on the content of the
input message.
-->
- <assign>
- <copy>
- <from>
<error xmlns="http://services.otn.com">Bankruptcy Report</error>
</from>
<to variable="fault" part="payload"/>
</copy>
</assign>
<!--
Asynchronous callback to the requester.
Note: the callback location and correlation id is transparently
handled using WS-addressing.
-->
<reply name="replyOutput" partnerLink="client" portType="tns:CreditRatingService"
operation="process" variable="fault" faultName="tns:NegativeCredit"/>
</sequence>
</case>
- <otherwise>
- <sequence>
- <!--

```

Appendices

This part describes Oracle SOA Suite appendices.

This part contains the following appendices:

- [Appendix A, "Building a Custom Worklist Client"](#)

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 A.1, "Introduction to Building Clients for Workflow Services"](#)
- [Section A.2, "Packages and Classes for Building Clients"](#)
- [Section A.3, "Workflow Service Clients"](#)
- [Section A.4, "Classpaths for Clients Using SOAP"](#)
- [Section A.5, "Classpaths for Clients Using Remote EJBs"](#)
- [Section A.6, "Classpaths for Clients Using Local EJBs"](#)
- [Section A.7, "EJB References in Web Applications"](#)
- [Section A.8, "Initiating a Task"](#)
- [Section A.9, "Writing a Worklist Application Using the HelpDeskUI Sample"](#)

A.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 A-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 A-1 Building a Client for Workflow Services—Setting the Outcome to Approved

```

try
{
//Create JAVA WorkflowServiceClient
IWorkflowServiceClient wfSvcClient =
    WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory.JAVA_CLIENT);
//Get the task query service
ITaskQueryService querySvc = wfSvcClient.getTaskQueryService();

//Login as jstein
IWorkflowContext ctx = querySvc.authenticate("jstein",
    "welcome1",
    null, //Use default realm
    null);//Not logging in on behalf of another user

//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());
}

```

A.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`
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 task metadata service.
- `oracle.bpel.services.workflow.task.ITaskService`
The interface for task service.
- `oracle.bpel.services.workflow.task.IRoutingSlipCallback`
The interface for callback class to receive callbacks during task processing.
- `oracle.bpel.services.workflow.task.IAssignmentService`
The interface for the assignment service.

A.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 EJB
- Facilitates changing from using one particular invocation mechanism to another, for example from SOAP/HTTP to remote EJB

The following class is used to create instances of the `IWorkflowServiceClient` interface:

```
oracle.bpel.services.workflow.client.WorkflowServiceClientFactory
```

`WorkflowServiceClientFactory` has a number of 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 EJB interface to invoke the workflow services.
- `WorkflowServiceClientFactory.REMOTE_CLIENT`—The client uses a remote EJB 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 allow you to specify the connection properties directly (rather than having the factory load them from the `wf_client_config.xml` file), and allow you to specify a logger to use to log client activity.

The following enhancements to the workflow service clients are included in this release:

- In addition to specifying the connection information necessary for the clients to talk to the server in the `wf_client_config.xml` file, you can also pass in a map (instead of using the `wf_client_config.xml` file) with the necessary properties. This is achieved by using a different method on the `WorkflowServiceClientFactory`, as shown in [Example A-2](#) and [Example A-3](#).

Example A-2 Method for Remote EJB Clients

```
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.EJB_PROVIDER_URL,
"ormi://localhost/soa-infra");

properties.put(IWorkflowServiceClientConstants.CONNECTION_PROPERTY.EJB_SECURITY_CREDENTIALS,
"welcome1");

properties.put(IWorkflowServiceClientConstants.CONNECTION_PROPERTY.EJB_SECURITY_PRINCIPAL,
"oc4jadmin");

IWorkflowServiceClient client =
WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory.REMOTE_CLIENT,
properties, null);
```

Example A-3 Method for SOAP Clients

```
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 A-4](#) shows how a logger can be passed to the workflow service clients.

Example A-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. [Table A-1](#) shows the clients available for each of the services.

Table A-1 Clients Available for the Workflow Services

Service Name	Supports SOAP Web Services	Supports Remote EJB	Supports Local EJB
Task Service	Yes	Yes	Yes
Task Query Service	Yes	Yes	Yes
Task Metadata Service	Yes	Yes	Yes
Task Reports Service	Yes	Yes	No
User Metadata Service	Yes	Yes	Yes
Runtime Config Service	Yes	Yes	Yes
Evidence Store Service	Yes	Yes	Yes
Identity Service:	-	-	-
■ BPM Authentication Service	Yes	No	No
■ BPM Authorization Service	Yes	No	No

The client classes use the configuration file `wf_client_config.xml` for the service end points. In the client classpath, this file is in the classpath directly, meaning the containing directory is in the classpath. The `wf_client_config.xml` file contains:

- A section for EJB configuration

```
<ejb>
  <serverURL>ormi://localhost/hw_services</serverURL> <!-- for stand alone -->
  <!--serverURL>opmn:ormi://localhost:home/hw_services</serverURL--> <!-- for
  opmn managed instance -->
  <user>oc4jadmin</user>
  <password>welcome1</password>
```

```
<initialContextFactory>oracle.j2ee.rmi.RMIInitialContextFactory</initialContextFactory>
```

```
</ejb>
```

- A section for SOAP end points for each of the services

```
<taskService>
<soapEndPoint>http://localhost:8888/integration/services/TaskService/
  TaskServicePort</soapEndPoint>
</taskService>
```

A.3.1 The IWorkflowServiceClient Interface

The `IWorkflowServiceClient` interface provides methods, summarized in [Table A-2](#), for obtaining handles to the various workflow services interfaces.

Table A-2 *IWorkflowServiceClient Methods*

Method	Interface
<code>getTaskService</code>	<code>oracle.bpel.services.workflow.task.ITaskService</code>
<code>getTaskQueryService</code>	<code>oracle.bpel.services.workflow.query.ITaskQueryService</code>
<code>getTaskReportService</code>	<code>oracle.bpel.services.workflow.report.ITaskReportService</code>
<code>getTaskMetadataService</code>	<code>oracle.bpel.services.workflow.metadata.ITaskMetadataService</code>
<code>getUserMetadataService</code>	<code>oracle.bpel.services.workflow.user.IUserMetadataService</code>
<code>getRuntimeConfigService</code>	<code>oracle.bpel.services.workflow.runtimeconfig.IRuntimeConfigService</code>
<code>getTaskEvidenceService</code>	<code>oracle.bpel.services.workflow.metadata.ITaskMetadataService</code>

A.4 Classpaths for Clients Using SOAP

SOAP clients must have the following JAR files in their classpath:

- `orasaa.jar`
- `xmlparserv2.jar`
- `xml.jar`
- `soap.jar`
- `saa-api.jar`
- `oc4jclient.jar`
- `bpm-infra.jar`
- `bpm-services.jar`
- `bpm-services-client.jar` (only if you are using the ADF data controls for workflow)

An alternative is to use the `wsclient_extended.jar` file, which is available for download from Oracle Technology Network at

http://download.oracle.com/otn/java/oc4j/1013/wsclient_extended.zip

See the chapter "Web Service Client APIs and JARs" in the section "Simplifying the Classpath with `wsclient_extended.jar`" in *Oracle Application Server Web Services Developer's Guide 10g Release 3 (10.1.3)*, at

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

Using `wsclient_extended.jar` simplifies the classpath to include the following JAR files:

- `wsclient_extended.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 classpath.

A.5 Classpaths for Clients Using Remote EJBs

Clients using remote EJBs must have the following JAR files in their classpath:

- `xmlparserv2.jar`
- `xml.jar`
- `oc4jclient.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 classpath.

A.6 Classpaths 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 classpath, either because they are part of `soa-infra` or because they inherit the classpath as the child of `soa-infra`.

Note: Client applications no longer use the `system\services\config` or `system\services\schema` directories in the classpath.

A.7 EJB 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 EJB local references in its `web.xml` file. The local references for each of the services are shown in [Example A-5](#) and [Example A-6](#).

Example A-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 A-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 28, "Human Task Services,"](#) for more information on `TaskQueryService`, `TaskReportService`, `UserMetadataService`, and `RuntimeConfigService`.

A.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`

A.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.

A.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 A-7](#) shows how to set a task payload.

Example A-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.

A.8.3 Initiating a Task Programmatically

[Example A-8](#) shows how to initiate a vacation request task programmatically.

Example A-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;

```

A.9 Writing a Worklist Application Using the HelpDeskUI Sample

The following example shows how to modify the help desk interface that is part of the HelpDeskServiceRequest demo found at

SOA_Oracle_home\bpel\samples\demos\HelpDeskServiceRequest\HelpDeskUI

To write a worklist application

1. Create the workflow context by authenticating the user.

```

// get workflow service client
IWorkflowServiceClient wfSvcClient =
    WorkflowServiceClientFactory.getWorkflowServiceClient
        (WorkflowServiceClientFactory.JAVA_CLIENT);

//get the workflow context
IWorkflowContext wfCtx =
    wfSvcClient.getTaskQueryService().authenticate(userId, pwd,
    oracle.tip.pc.services.identity.config.ISConfiguration.getDefaultRealmName(),
    null);

```

This is Step 3 in [Section A.1, "Introduction to Building Clients for Workflow Services."](#)

The `login.jsp` file of `HelpDeskServiceRequest` 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 A-9](#) shows sample code from the `login.jsp` file.

Example A-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.JAVA_CLIENT);
                IWorkflowContext wfCtx =
                    wfSvcClient.getTaskQueryService().authenticate(userId, pwd,
                        oracle.tip.pc.services.identity.config.ISConfiguration.getDefaultRealmName(), 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>
</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.ASSIGNMENT_FILTER_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 A.1, "Introduction to Building Clients for Workflow Services."](#)

The `statusPage.jsp` file of `HelpDeskServiceRequest` is used to display the status of help desk requests. [Example A-10](#) shows the `statusPage.jsp` example code.

Example A-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.JAVA_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>
```


A

- actionable notification
 - definition, 23-1
- actional e-mails, 26-56, 28-24
- activities
 - definition, 11-12
 - overview, 11-12
- activity sensors
 - definition, 24-1
- Adapter Configuration wizard
 - starting, 11-18
- adapters
 - configuring, 11-18
 - definition, 11-18
 - in Oracle JDeveloper, 11-18
 - service names, 11-19
- adding a cross reference table column, 10-6
- adding columns to domain value maps, 9-9
- adding rows to domain value maps, 9-9
- ADF bindings
 - files for, 35-5
- ADF Model layer, introduced, 35-1
- ADF task flow for human tasks, 27-2
- advanced formatting, message sources, 36-4
- aggregate functions in calculations, 32-4
- alerts
 - history, 34-7
- ant
 - using the developer prompt, 11-2
- Application Navigator
 - contents of, 11-5
 - definition, 11-5
 - location of in Oracle JDeveloper, 11-4
- arrays
 - determining the size of, 12-22
 - in transformations, 4-23
 - manipulating, 12-20
 - maxOccurs attribute, 12-20
 - SOAP-encoded arrays not supported, 12-25
 - statically indexing into, 12-21
- assert facts and execute rule set operation
 - definition, 25-16
- assert facts only operation
 - definition, 25-16
- assert facts, execute rule set, and retrieve results
 - operation
 - definition, 25-17
- assert facts, execute rule set, retrieve results, and reset the session operation
 - definition, 25-17
- assertion tests
 - overview, 43-2
- assertions
 - creating value asserts, 43-17
 - creating XML asserts, 43-19
 - in BPEL test suites, 43-5
- assign activity
 - adding to an asynchronous service, 14-13
 - copying data, 12-10
 - description, 12-2
 - for data manipulation, 12-2
 - formatting the e-mail message body as HTML, 23-9
 - in asynchronous services, 14-13
- assignment service
 - configuration, 28-30
 - deploying a custom assignment service, 28-36
 - dynamic assignment functions, 28-30, 28-31, 28-32
 - dynamically assigning task participants, 28-32
 - example of implementation, 28-34
 - implementing, 28-33
 - overview, 28-33
- asynchronous callbacks, 14-3
- asynchronous interaction with a notification timer
 - BPEL process as the client, 22-5
 - BPEL process as the service, 22-5
 - definition, 22-4
- asynchronous interaction with timeout
 - BPEL process as the client, 22-4
 - BPEL process as the service, 22-4
 - definition, 22-4
- asynchronous interactions
 - BPEL process as the client, 22-3
 - BPEL process as the service, 22-3
 - definition, 22-3
 - returning faults, 17-7
- asynchronous processes
 - dehydration store, 14-10
 - using dehydration, 14-10
- asynchronous services

- assign activities, 14-13
- calling, 14-4, 14-10
- correlating messages, 14-6
- correlation IDs, 14-6
- correlation sets, 14-14
- invoke activities, 14-5, 14-11
- managing multiple instances, 14-6
- parallel flows, 15-1
- partner links, 14-4, 14-10
- partnerLinkTypes, 14-3
- receive activities, 14-5, 14-13
- reply activities, 14-10
- use case, 14-1
- WS-Addressing, 14-6
- attachments
 - sending with the notification wizard, 23-6
 - task attachments with e-mail notifications, 26-57, 28-27
 - using style sheets, 26-59
 - using WordML style sheets, 26-59
- attribute labels
 - internationalization, 28-15
- attributes
 - manipulating, 12-14
- auto mapping
 - in transformations, 4-25
 - with confirmation in transformations, 4-27

B

- batching
 - message batching limitations with Oracle Business Activity Monitoring, 24-19
- bindingFault
 - definition, 17-4
- Boolean values
 - assigning, 12-13
- BPEL design environment
 - overview, 11-2
- BPEL files
 - definition, 11-5
 - partner links definition, 13-2
- BPEL processes
 - common interaction patterns, 22-1
 - invoking with a Web Service/SOAP interface, 20-2
 - sending messages from a Java/JSP application, 20-1
- BPEL projects
 - naming conventions, 11-3
- BPEL test
 - assertions overview, 43-2
 - creating test suites, 43-6
 - creating value asserts, 43-17
 - creating XML asserts, 43-19
 - definition, 43-1
 - deploying test suites, 43-21
 - emulating inbound messages, 43-9
 - emulations overview, 43-2
 - naming limitations on test suites and test

- cases, 43-6
- process code coverage overview, 43-3
- running test suites from Oracle BPEL Control, 43-22
- simple value assert, 43-2
- test case overview, 43-2
- test suite assertions, 43-5
- test suite components, 43-3
- test suite emulations, 43-4
- test suites overview, 43-2
- test suites process initiation, 43-3
- viewing test results, 43-22
- XML assert, 43-3
- BPEL XPath functions
 - examples, 12-3
- BPEL_ACTIVITY_SENSOR_VALUES
 - sensor public view, 24-21
- BPEL_FAULT_SENSOR_VALUES
 - sensor public view, 24-21
- BPEL_PROCESS_INSTANCES
 - sensor public view, 24-20
- BPEL_VARIABLE_SENSOR_VALUES
 - sensor public view, 24-22
- bpelx exec extension
 - for embedding Java code in a BPEL process, 18-2
- bpelx:append extension
 - description, 12-15
- bpelx:copyList extension
 - description, 12-19
- bpelx:exec extension
 - built-in methods, 18-4
- bpelx:headerVariable extension
 - description, 12-27
- bpelx:insertAfter extension
 - description, 12-16
- bpelx:insertBefore extension
 - description, 12-15
- bpelx:remove extension
 - description, 12-17
- bpelx:rename extension
 - description, 12-18
- bpelx:validate extension
 - description, 12-20
- building expression with domain value map
 - functions, 9-14
- business faults
 - definition, 17-3
- business rules
 - definition, 25-1
 - deployment and run time, 25-19
 - integrating with BPEL processes through the decision service wizard, 25-14
 - use case for data validation and constraint checks, 25-13
 - use case for dynamic processing, 25-13
 - use case for externalizing decision points in the process, 25-13
 - use case for human workflow, 25-13
 - use cases, 25-13
 - with Oracle BPEL Process Manager, 25-2

business rules dictionary
definition, 25-3

C

calculated fields, 32-4
calculations
 aggregate functions, 32-4
 datetime functions, 32-4
 expressions, 32-4
 string functions, 32-4
callback, 6-19
callback classes
 specifying on task status, 26-61
callbacks
 task routing and customization in BPEL
 callbacks, 26-62, 26-79
 viewing, 26-75
catch branch
 fault handling, 17-7
channels
 e-mail, 23-5
 SMS, 23-9
 voice mail, 23-10
class names
 specifying in the external routing service
 participant type, 26-39
classpaths
 for clients using local EJBs, A-7
 for clients using remote EJBs, A-7
 for clients using SOAP, A-6
clearing data objects, 32-15
code coverage tests
 overview, 43-3
compensate activity
 definition, 17-8
 fault handling, 17-8
complex type
 variables, 12-10
Component Palette
 definition, 11-9
 location of in Oracle JDeveloper, 11-4
 transformation functions, 11-9
components
 of Oracle SOA Suite, 1-3
composite
 overview, 2-4
concat function
 description, 12-12
conditional branching logic
 definition, 16-1
 use case, 16-1
 use of XPath expressions, 16-1
 using switch activities, 16-2
 using while activities, 16-4
conditional processing
 with xsl choose, 4-22
 with xsl if, 4-21
constant values
 in transformations, 4-14

contents, data object, 32-7
copying security filters, 32-12
core XPath functions
 examples, 12-3
correlation ID
 WS-Addressing, 14-6
correlation sets
 associating with receive activities, 14-22
 creating, 14-21
 creating property aliases, 14-23
 definition, 14-7, 14-9
 tutorials, 14-14
 WSDL file content, 14-25
correlations, 14-6
countNodes function, 12-22
create domain value maps, 9-4
create instance
 definition, 14-5
 in receive activities, 14-5
createInstance attribute, 14-6
creating cross reference tables, 10-4
creating folders for data objects, 32-8
creating mediator component
 mediator files, 5-3
cross reference table look up, 10-13
 xref
 lookupXRef function, 10-13
 lookupXRef1M function, 10-16
cross reference tables, 10-1
 adding a column, 10-6
 creating, 10-4
 deleting a column, 10-7
 deleting values, 10-17
 looking up, 10-13
 modifying, 10-4
 populating columns, 10-7
 xref
 lookupXRef function, 10-13
 lookupXRef1M function, 10-16
 markForDelete function, 10-17
 populateXRefRow function, 10-7
 populateXRefRow1M function, 10-12
cross references
 creating, 10-4
 introduction, 10-1
 modifying, 10-4
 overview, 10-1
 schema definition file, 10-20
custom
 sensor publish type, 24-2
custom escalation function
 using, 28-36
custom plug-ins
 use with identity service, 28-10

D

data controls
 creating, 35-4
 displayed on the Data Controls panel, 35-5

- Data Controls panel
 - icons defined, 35-5
 - using to create a user interface, 35-6
- data manipulation
 - accessing fields with complex type variables, 12-10
 - assigning Boolean values, 12-13
 - assigning date or time, 12-13
 - assigning literal strings, 12-12
 - assigning numeric values, 12-11
 - concatenating strings, 12-12
 - converting from a string to a structured XML object type, 12-25
 - copying data between variables, 12-10
 - determining array sizes, 12-22
 - dynamically indexing into a data sequence, 12-22
 - generating array-equivalent functionality with the `genEmptyElem` function, 12-24
 - initializing variables, 12-9
 - manipulating arrays, 12-20
 - manipulating attributes, 12-14
 - mathematical calculations with XPath functions, 12-11
 - statically indexing into a data sequence, 12-21
 - with assign activities, 12-2, 12-10
 - with XQuery and XSLT, 12-4
- data mapper
 - using, 6-32
- data object contents, 32-7
- data object permissions, 32-4
- data objects, 32-1
 - organizing, 32-8
- data objects, adding dimensions, 32-12
- data objects, clearing, 32-15
- data objects, creating folders, 32-8
- data objects, defining, 32-1
- data objects, deleting, 32-15
- data objects, general information, 32-6
- data objects, moving, 32-14
- data objects, renaming, 32-14
- data objects, security filters, 32-10
- data sequences
 - dynamically indexing into, 12-22
- database
 - sensor publish type, 24-2
- date time stamp field, 32-4
- dates
 - assigning, 12-13
- datetime functions in calculations, 32-4
- decide activity
 - assert facts and execute rule set operation
 - definition, 25-16
 - assert facts only operation definition, 25-16
 - assert facts, execute rule set, and retrieve results operation definition, 25-17
 - assert facts, execute rule set, retrieve results, and reset the session operation definition, 25-17
 - definition, 25-16
 - execute function and reset the session operation definition, 25-17
 - execute function operation definition, 25-17
 - retrieve results operation definition, 25-17
- decision service
 - architecture, 25-3
 - configuration file contents, 25-4
 - definition, 25-2
 - deployment and run time, 25-19
- decision service wizard
 - for integrating business rules with BPEL processes, 25-14
- defining a fault handler, 17-5
- dehydration
 - definition, 14-10
- dehydration store, 14-10
- deleting a cross reference table column, 10-7
- Deleting a Mediator, 5-22
- deleting a routing service, 5-22
- deleting cross reference table value, 10-17
 - xref
 - `markForDelete` function, 10-17
- deleting data objects, 32-15
- deleting folders, 32-10
- demos
 - binding faults, 17-5
 - run-time faults, 17-5
 - sensor actions, 24-3
 - sensors, 24-3
- deployment
 - from Oracle JDeveloper, 3-2
- developer prompt
 - required when deploying services through ant or obant, 11-2
- Diagram window, 11-5
 - definition, 11-5
 - location of in Oracle JDeveloper, 11-4
- dictionaries
 - in transformations, 4-29
- digital signatures, 28-16
- dimensions, adding to data objects, 32-12
- dimensions, time, 32-13
- domain value maps
 - add columns, 9-9
 - add rows, 9-9
 - creation, 9-4
 - dvm
 - `lookupValue` function, 9-9
 - `lookupValue1M` function, 9-10
 - editing, 9-8
 - features, 9-1
 - introduction, 9-1
 - one-to-many mapping, 9-3
 - overview, 9-1
 - qualifier order, 9-2
 - qualifiers, 9-2
 - reordering the columns, 9-9
 - using, 9-9
 - using in a transformation, 9-11
 - using `lookupValue` functions, 9-14
- domain value maps features, 9-1
 - one-to-many mapping, 9-3

- qualifier order, 9-2
- qualifiers, 9-2
- domain value maps functions
 - dvm
 - lookupValue, 9-9
 - lookupValue1M, 9-10
- domain value maps qualifiers, 9-2
- domain.xml file
 - increasing the syncMaxWaitTime property, 19-4
 - location, 19-4
- drop handlers, custom for task display form, 27-5
- drop handlers, standard ADF, 27-10
- dvm
 - lookupValue function, 9-9
 - lookupValue1M function, 9-10
- dynamic assignment
 - of task participants, 26-26, 26-29, 26-33, 26-36, 26-38
- dynamic assignment functions
 - configuring, 28-31
 - configuring display names, 28-32
 - definition, 28-30
 - implementing, 28-31

E

- edit domain value maps, 9-8
 - add columns, 9-9
 - add rows, 9-9
 - reordering the columns, 9-9
- EJB
 - security in EJB Web services, 28-3
 - support in workflow services, 28-1
- elements
 - ignoring in XSLT documents, 4-32
- e-mail
 - dynamically setting addresses, 23-11
 - making e-mails actionable, 26-56, 28-24
 - notifications support, 23-2, 23-5
- e-mail attachments
 - notifications support, 23-6
- e-mail messages
 - HTML content for message body, 23-9
- empty activity
 - definition, 17-8
 - fault handling, 17-8
- emulation tests
 - overview, 43-2
- emulations
 - emulating inbound messages, 43-9
 - in BPEL test suites, 43-4
- ending
 - tasks, 26-46
- enterprise message sources, 36-1
- enterprise message sources, copying, 36-10
- enterprise message sources, defining, 33-2, 36-2
- enterprise message sources, deleting, 33-2, 36-10
- enterprise message sources, editing, 33-2, 36-10
- entity variable
 - using, 12-4

- escalating
 - tasks, 26-46
- escalation policy
 - escalate after, 26-51
 - overview, 26-47
 - specifying, 26-58
- evaluation time
 - definition, 24-4
- evidence store, 28-16
- evidence store service, 28-2, A-5
- example code, 36-5
- exceptions, 17-3
- execute function and reset the session operation
 - definition, 25-17
- execute function operation
 - definition, 25-17
- expiration policy
 - expire after, 26-49
 - never expire, 26-49
 - overview, 26-47
 - renew after, 26-50
- export file sample, ICommand, 37-18
- expression builder dialog
 - using domain value map functions, 9-14
- expression constants
 - variable initialization, 12-9
- expressions in calculations, 32-4
- external routing service
 - configuring, 26-38
 - definition, 26-11, 26-38
 - specifying the class name, 26-39
 - workflow participant type, 26-11, 26-38

F

- facades
 - See* XML facades
- fault bindings, 7-4
- fault handling, 6-19, 17-5
 - binding faults demo, 17-5
 - defining, 17-1, 17-5
 - importing RuntimeFault.wsdl, 17-5
 - modifying the WSDL files, 17-5
 - returning external faults, 17-7
 - throwing internal faults, 17-6
 - use case, 17-1
 - using catch branches, 17-7
 - using compensate activities, 17-8
 - using empty activities, 17-8
 - using scope activities, 17-6, 17-7
 - using terminate activities, 17-9
 - using the getFaultAsString function, 17-5
 - using throw activities, 17-6
- fault policies, 7-1
 - actions, 7-4
 - component level, 7-4
 - composite level, 7-4
 - conditions, 7-2
- fault sensors
 - definition, 24-2

- fault-bindings.xml, 7-9
- fault-policies.xml, 7-5
- faults
 - categories of faults in BPEL, 17-3
 - Qname fault name, 17-3
 - returning external faults, 17-7
 - standard faults, 17-3
 - throwing internal faults, 17-6
- fields, calculated, 32-4
- fields, lookup, 32-3
- fields, timestamp, 32-4
- filters, copying, 32-12
- filters, security, 32-10
- fire and forget
 - one-way message, 22-1
- flat query, 35-6
- flowN activity
 - definition, 15-2
- folder permissions, 32-9
- folders, deleting, 32-10
- folders, renaming, 32-10
- functions
 - chaining in transformations, 4-16
 - concat, 12-12
 - countNodes, 12-22
 - custom, 12-4
 - deprecated workflow service and identity service functions, 28-37
 - descriptions, 4-15
 - dynamically setting e-mail addresses and telephone numbers, 23-11
 - editing in transformations, 4-16
 - editing XPath expressions in transformations, 4-19
 - examples, 12-3
 - functions prefixed with xp20 or orcl, 4-15
 - genEmptyElem, 12-24
 - getCurrentDate, 12-13
 - getCurrentDateTime, 12-13
 - getCurrentTime, 12-13
 - getFaultAsString, 17-5
 - getUserProperty, 23-11
 - getVariableData, 23-11
 - in transformations, 4-15
 - location of function descriptions, 12-4
 - mimic XPath 2.0 standards, 28-36
 - parseEscapedXML, 12-25
 - position, 12-21
 - prefixed with xp20 or orcl, 4-15
 - processXSLT, 23-9
 - readFile, 23-8
 - selecting an data sequence element, 12-21
 - workflow related, 28-36
- FYI assignee
 - configuring, 26-37
 - definition, 26-11, 26-37
 - workflow participant type, 26-11, 26-37

G

- genEmptyElem function
 - description, 12-24
- getCurrentDate function
 - description, 12-13
- getCurrentDateTime function
 - description, 12-13
- getCurrentTime function
 - description, 12-13
- getFaultAsString function
 - description, 17-5
- getUserProperty function
 - example, 23-11
- getVariableData function
 - description, 12-12
 - example, 23-11
 - using in mathematical calculations, 12-11
- global task variable name
 - specifying in human task activities, 26-78
- group query, 35-6
- group vote
 - configuring, 26-27
 - consensus percentage, 26-30
 - default outcome, 26-30
 - definition, 26-11, 26-27
 - immediately triggering a voted outcome when a minimum percentage is met, 26-30
 - specifying group voting details, 26-30
 - waiting until all votes are in before triggering an outcome, 26-30
 - workflow participant type, 26-11, 26-27

H

- headers
 - SOAP headers, 12-27
- heap size
 - increasing, 4-39
- human task activity
 - associating with a BPEL process, 26-69
 - identification key, 26-78
 - including the task history of other tasks, 26-78
 - scope name and global task variable name, 26-78
 - specifying a task initiator and task priority, 26-72
 - specifying a task title, 26-71
 - specifying task parameters, 26-72
 - task owner, 26-78
 - viewing BPEL callbacks, 26-75
- human task definition
 - associating with a BPEL process, 26-12
- Human Task editor
 - abruptly completing a condition, 26-41
 - actional e-mails, 26-56, 28-24
 - allowing all participants to invite other participants, 26-40
 - assigning task participants by name or expression, 26-25, 26-29, 26-32, 26-35, 26-37
 - bypassing task participants, 26-26, 26-30, 26-33, 26-36
 - definition, 26-2

- dynamically assigning task participants by
 - expression, 26-26, 26-29, 26-33, 26-36, 26-38
- editing notification messages, 26-55
- escalate after policy, 26-51
- escalating, renewing, or ending a task, 26-46
- escalation and expiration policy overview, 26-47
- escalation rules, 26-58
- expire after policy, 26-49
- external routing service task participant, 26-38
- FYI assignee task participant, 26-37
- group vote task participant, 26-27
- group voting details, 26-30
- inviting additional task participants, 26-27, 26-34, 26-36
- management chain task participant, 26-31
- multilingual settings, 26-59, 28-23
- never expire policy, 26-49
- notification preferences, 26-53
- notifying recipients of changes to task status, 26-54
- number of task approvers, 26-33
- overriding default exception management, 26-60
- renew after policy, 26-50
- securing notifications, 26-56, 28-28
- sequential list of approvers task participant, 26-34
- setting up reminders, 26-55
- sharing attachments and comments with task participants, 26-30, 26-38
- single approver task participant, 26-24
- specifying callback classes, 26-61
- specifying class names, 26-39
- style sheets in attachments, 26-59
- task attachments with e-mail notifications, 26-57, 28-27
- task outcome, 26-17
- task owner specification through the user directory, 26-18
- task owner specification through XPath expressions, 26-20
- task participants, 26-22
- task payload data structure, 26-21
- task routing and customization in BPEL callbacks, 26-62, 26-79
- task title and priority, 26-16
- time limits for acting on tasks, 26-27, 26-31, 26-34, 26-36
- WordML style sheets in attachments, 26-59

I

- ICommand
 - regular expressions, 37-19
 - sample export file, 37-18
- ICommand utility, 37-1
- ICommand, detailed command descriptions, 37-6
- ICommand, general command and option syntax, 37-2
- ICommand, summary of commands, 37-5
- identification key
 - specifying in human task activities, 26-78
- identity service
 - definition, 26-8, 28-8
 - deprecated functions, 28-37
 - determining a user's local language and time zone, 29-45
 - EJB, SOAP, and Java support, 28-2, A-5
 - providers, 28-10
 - support for in workflows, 28-8
 - supported task operations, 28-8
 - use with custom plug-ins, 28-10
 - use with JAZN, 28-8, 28-10
 - use with LDAP, 28-8, 28-10
 - WSDL file location, 28-2
 - XPath extension functions, 28-36
- import
 - source and target schemas into a transformation, 4-3
 - two schema files of the same name into the same project is not supported, 11-12
- indexes, in data objects, 32-14
- indexing methods
 - using XPath, 12-22
- instances
 - starting new, 14-6
- interaction patterns
 - asynchronous interaction with notification timer, 22-4
 - asynchronous interaction with timeout, 22-4
 - asynchronous interactions, 22-3
 - common patterns between a BPEL process and another application, 22-1
 - multiple interactions, 22-9
 - one request, a mandatory response, and an optional response, 22-7
 - one request, multiple responses, 22-5
 - one request, one of two possible responses, 22-6
 - one-way message, 22-1
 - partial processing, 22-8
 - synchronous interactions, 22-2
- invocation patterns
 - assert facts and execute rule set operation definition, 25-16
 - assert facts only operation definition, 25-16
 - assert facts, execute rule set, and retrieve results operation definition, 25-17
 - assert facts, execute rule set, retrieve results, and reset the session operation definition, 25-17
 - execute function and reset the session operation definition, 25-17
 - execute function operation definition, 25-17
 - retrieve results operation definition, 25-17
- invoke activity, 14-5
 - adding to an asynchronous service, 14-11
 - definition, 13-2
 - in asynchronous services, 14-5, 14-11
 - in synchronous services, 13-2, 13-4

J

- Java
 - support in workflow services, 28-1
- Java applications
 - wrapped as SOAP services, 18-1
- Java embedding
 - bpel:exec extension, 18-4
 - embedding code in a BPEL process, 18-2
 - example, 18-5
 - in a BPEL process, 18-1
 - using bpelx exec, 18-2
- Javadocs
 - location of, 28-11
- Java/JSP applications
 - calling a BPEL process, 20-1
- JAZN
 - storing a user's local language and time zone, 29-45
 - use with identity service, 28-8, 28-10
- JMS adapter
 - sensor publish type, 24-2
- JMS Queue
 - sensor publish type, 24-2
- JMS Topic
 - sensor publish type, 24-2
- JUnit
 - BPEL test results, 43-22

L

- languages
 - changing, 29-46
 - preferences, 29-45
 - setting in JAZN, 29-45
 - setting in LDAP, 29-45
- layouts, data object, 32-7
- LDAP
 - storing a user's local language and time zone, 29-45
 - used with identity service, 28-8, 28-10
- literal strings
 - assigning, 12-12
- literal XML
 - variable initialization, 12-9
- localization, worklist, 29-45
- log files
 - viewing, 3-11
- Log window
 - definition, 11-12
 - location of in Oracle JDeveloper, 11-4
- logging
 - BPEL process properties, 3-14
- looking up cross reference tables, 10-13
 - xref
 - lookupXRef function, 10-13
 - lookupXRef1M function, 10-16
- lookup fields, 32-3
- lookupValue functions
 - dvm
 - lookupValue function, 9-9

lookupValue1M function, 9-10

M

- management chain
 - configuring, 26-31
 - definition, 26-11, 26-31
 - highest title of approver, 26-34
 - maximum number of chain levels up, 26-33
 - workflow participant type, 26-11, 26-31
- map parameters
 - creating in transformations, 4-29
- map variables
 - creating in transformations, 4-29
- master and detail process coordinations, 21-1
- maxOccurs attribute, 12-20, 12-22
 - setting for transformations, 4-39
- mediator component
 - mediator files, 5-3
- Mediator Creation
 - Specifying Operation or Event Subscription Properties, 5-19
- Mediator Designer Environment
 - Application Navigator, 5-2
 - History Window, 5-4
 - Log Window, 5-4
 - Mediator Editor, 5-3
 - Property Inspector, 5-4
 - Source View, 5-4
 - Structure Window, 5-4
- Mediator Editor, 5-3
- Mediator Files
 - .componentType, 5-3
 - Composite.xml, 5-3
 - .mplan, 5-3
 - .wsdl, 5-3
- message source advanced formatting, 36-4
- message source example code, 36-5
- message sources, 36-1
- minOccurs attribute
 - setting for transformations, 4-40
- modes
 - xref
 - populateXRefRow function, 10-7
 - populateXRefRow1M function, 10-12
- Modifying a Mediator, 5-20
 - Modifying Event Subscriptions, 5-21
 - Modifying Operations, 5-20
- modifying cross reference tables
 - adding a column, 10-6
 - deleting a column, 10-7
- Modifying Mediator Event Subscriptions, 5-21
- Modifying Mediator Operations, 5-20
- multilingual settings
 - specifying in tasks, 26-59, 28-23
- myRole attribute
 - definition, 14-5

N

- named templates
 - creating, 4-17
 - in functions, 4-17
- naming conventions
 - for BPEL projects, 11-3
- NLS
 - configuration overview, 28-38
- notification messages
 - editing, 26-55
- notification services
 - actionable e-mails, 28-24
 - configuring the notification channel, 28-21
 - definition, 26-8, 28-11
 - limitations on setting validateXML to true, 23-13
 - multilingual settings, 28-23
 - notification contents, 28-22
 - reliability support, 28-27
 - sending inbound and outbound
 - attachments, 28-27
 - sending inbound comments, 28-27
 - sending reminders, 28-28
 - sending secure notifications, 28-28
 - specifying participant notification preferences, 26-53
 - WSDL file location, 28-2
- notifications
 - configuring in Oracle JDeveloper, 23-3
 - definition, 23-1, 26-2
 - dynamically setting e-mail addresses and telephone numbers, 23-11
 - e-mail attachment support, 23-6
 - e-mail support, 23-2, 23-5
 - formatting the e-mail message body as HTML, 23-9
 - number of retries, 23-3
 - number of retry intervals, 23-3
 - reliable notification service, 23-3
 - selecting recipients by browsing the user directory, 23-12
 - SMS support, 23-9
 - use case, 23-1
 - using Oracle Application Server Wireless, 23-2
 - voice mail support, 23-10
- notifications and reminders
 - in tasks, 28-20
- numeric values
 - assigning, 12-11

O

- obant
 - using the developer prompt, 11-2
- onAlarm branch
 - of pick activity, 19-2
- one-to-many mapping, 9-3
- onMessage branch
 - of pick activity, 19-2
- Oracle Application Server Wireless
 - wireless and voice component, 23-2

- Oracle BAM
 - See Oracle Business Activity Monitoring
- Oracle BAM Server
 - creating a BPEL sensor, 24-17
 - creating a BPEL sensor action, 24-17
 - creating a connection to, 24-16
- Oracle BPEL Server
 - overview, 11-18
- Oracle BPEL Worklist Application
 - responding to tasks from, 26-1
- Oracle Business Activity Monitoring
 - creating a BPEL sensor action for Oracle BAM Server, 24-17
 - creating a BPEL sensor for Oracle BAM Server, 24-17
 - creating a connection to Oracle BAM Server, 24-16
 - definition
 - integration with Oracle BPEL Process Manager sensors, 24-15
 - message batching limitations, 24-19
 - overview, 24-15
- Oracle Business Rules RL Language
 - definition, 25-3
- Oracle Business Rules Rule Author
 - definition, 25-3
- Oracle Business Rules Rules Engine
 - definition, 25-3
- Oracle Business Rules SDK
 - definition, 25-3
- Oracle Enterprise Service Bus
 - overview, 1-4
- Oracle Internet Directory
 - storing a user's local language and time zone, 29-45
- Oracle JDeveloper, 11-5, 11-7
 - adapters, 11-18
 - Application Navigator, 11-5
 - Component Palette, 11-9, 11-10
 - configuring notifications, 23-3
 - creating sensors, 24-2
 - deploying test suites, 43-21
 - location of Application Navigator, 11-4
 - location of Component Palette, 11-4
 - location of Diagram window, 11-4
 - location of Log window, 11-4
 - location of Process Activities, 11-4
 - location of Property Inspector, 11-4
 - location of Structure window, 11-4
 - Log window, 11-12
 - overview of design environment, 11-2
 - Process Activities, 11-9
 - Property Inspector, 11-11
 - Services, 11-10
 - Structure window, 11-11
 - transformations, 4-1
- Oracle Mediator
 - define routing rules, 6-2
 - overview, 5-1
 - routing rules, 6-1

- Oracle Mediator component creation
 - mediator files, 5-3
- Oracle Mediator error handling
 - actions, 7-4
 - conditions, 7-2
 - fault bindings, 7-4
 - fault policies, 7-1
 - introduction, 7-1
 - using, 7-5
 - XML schema files, 7-5
- Oracle SOA Suite
 - components, 1-3
 - introduced, 1-2
- organizing data objects, 32-8
- overview, 24-1

P

- parallel flows
 - definition, 15-1
 - use case, 15-1
- parseEscapedXML function
 - description, 12-25
- partial processing
 - BPEL process as the client, 22-9
 - BPEL process as the service, 22-9
 - definition, 22-8
- participant types
 - external routing service, 26-11, 26-38
 - FYI assignee, 26-11, 26-37
 - group vole, 26-11, 26-27
 - management chain, 26-11, 26-31
 - sequential list of approvers, 26-11, 26-34
 - single approver, 26-10, 26-24
- partner links
 - adding to an asynchronous service, 14-10
 - BPEL file code example, 13-2
 - definition, 11-14, 13-2
 - displaying in Diagram window, 11-7
 - in asynchronous services, 14-4, 14-10
 - in synchronous services, 13-2
 - overview, 11-14
 - specifying a WSDL file, 11-14
- partnerLinkTypes
 - definition, 13-3, 14-3
 - in asynchronous services, 14-3
 - in synchronous services, 13-3
- partnerRole attribute
 - definition, 14-5
- patterns
 - of interaction between a BPEL process and another application, 22-1
- permissions, copying, 32-5
- permissions, data objects, 32-4
- permissions, setting on folders, 32-9
- pick activity
 - code example, 19-3
 - condition branches, 19-2
 - for timeouts, 19-1
 - onAlarm branch, 19-2

- onMessage branch, 19-2
- populating cross reference tables, 10-7
 - xref
 - populateXRefRow function, 10-7
 - populateXRefRow1M function, 10-12
- port types
 - definition, 13-4
 - in asynchronous services, 14-3
 - in synchronous services, 13-4
- ports
 - in synchronous services, 13-2
- position function
 - description, 12-21
- Priority property, 5-19
- Process Activities
 - definition, 11-9
 - location of in Oracle JDeveloper, 11-4
- process code coverage tests
 - overview, 43-3
- process initiation
 - in BPEL test suites, 43-3
- processes
 - naming conventions, 11-3
- processXSLT function
 - example, 23-9
- projects
 - BPEL file, 11-5
 - importing two schema files of the same name into
 - the same project is not supported, 11-12
 - in Application Navigator, 11-5
 - naming conventions, 11-3
 - WSDL file, 11-5
- property aliases
 - creating for correlation sets, 14-23
- Property Inspector
 - definition, 11-11
 - location of in Oracle JDeveloper, 11-4
- public views
 - BPEL_ACTIVITY_SENSOR_VALUES, 24-21
 - BPEL_FAULT_SENSOR_VALUES, 24-21
 - BPEL_PROCESS_INSTANCES, 24-20
 - BPEL_VARIABLE_SENSOR_VALUES, 24-22
 - sensors, 24-19
- publish types
 - creating a custom publisher, 24-8
 - custom, 24-2
 - database, 24-2
 - definition, 24-2
 - JMS Adapter, 24-2
 - JMS Queue, 24-2
 - JMS Topic, 24-2

Q

- Qname
 - fault name, 17-3
- qualifier, 9-2
 - qualifier order, 9-2
- qualifier order, 9-2

R

- readFile function
 - example, 23-8
- receive activity
 - adding to an asynchronous service, 14-13
 - associating with correlation sets, 14-22
 - create instance, 14-5
 - creating new instances, 14-6
 - in asynchronous services, 14-5, 14-13
- references
 - adding, 2-22
 - overview, 2-8
- regular expressions, ICommand, 37-19
- reliable notification service
 - persisting messages, 23-3
- reminders
 - for task notifications, 28-28
- remoteFault
 - definition, 17-5
- renaming data objects, 32-14
- renaming folders, 32-10
- renewing
 - tasks, 26-46
- reordering the columns in domain value maps, 9-9
- repeating elements
 - in transformations, 4-23
- replayFault
 - definition, 17-5
- reply activity
 - definition, 14-10
 - in asynchronous services, 14-10
- reporting schema
 - for database publish type of sensors, 24-20
- reports
 - correcting memory errors when generating for transformations, 4-39
 - customizing sample XML generation for transformations, 4-39
 - generating for transformations, 4-38
 - worklist, 29-41
- resource bundles
 - for displaying tasks in different languages, 26-59, 28-23
- Resource Palette
 - overview, 2-13
 - using, 2-19
- retrieve results operation
 - definition, 25-17
- roles
 - for partner links in asynchronous services, 14-4
- routing rules, 6-1
 - callback, 6-19
 - define, 6-2
 - defining, 6-2
 - fault handling, 6-19
 - filter expression, 6-7
 - introduction, 6-1
 - synchronous reply, 6-19
- routing services
 - deleting, 5-22

- .routing slip
 - definition, 26-2, 26-27
- RPC styles
 - differences with document-literal styles in WSDL files, 12-2, 12-26
- rules
 - worklist, 29-28
- rules engine
 - definition, 25-2
- runtime config service
 - definition, 26-9
 - EJB, SOAP, and Java support, 28-2, A-5
 - supported task operations, 28-13
 - WSDL file location, 28-3
- run-time exceptions, 17-3
- run-time faults
 - binding faults sample, 17-5
 - definition, 17-3
 - example, 17-5
- RuntimeFault.wsdl file
 - importing into a process, 17-5

S

- schema definition file
 - cross references, 10-20
- schema files
 - creating a transformation map file from imported schemas, 4-3
 - replacing in the XSLT Mapper, 4-32
- schemac
 - generating XML facades from WSDL or XSD files, 18-3
- scope activity
 - definition, 17-6
 - fault handling, 17-6, 17-7
- scope name
 - specifying in human task activities, 26-78
- SDO, 12-6
- security filters, copying, 32-12
- security filters, on data objects, 32-10
- security model
 - for workflow services, 28-3
 - in EJB Web services, 28-3
 - in SOAP Web services, 28-3
 - workflow context on behalf of a user, 28-4
- sensor actions
 - configuring, 24-6
 - creating a BPEL sensor action for Oracle BAM Server monitoring, 24-17
 - demos, 24-3
 - viewing definitions, 24-11
 - viewing metadata, 24-11
 - XSD schema file, 24-23
- sensor data
 - persisting in a reporting schema, 24-20
- sensorAction.xml file, 24-6, 24-7
- sensors, 24-1
 - activity sensors, 24-1
 - BPEL reporting schema, 24-20

- BPEL_ACTIVITY_SENSOR_VALUES public views, 24-21
- BPEL_FAULT_SENSOR_VALUES public views, 24-21
- BPEL_PROCESS_INSTANCES public views, 24-20
- BPEL_VARIABLE_SENSOR_VALUES public views, 24-22
- configuring, 24-3
- creating a BPEL sensor for Oracle BAM Server to monitor, 24-17
- creating a connection to Oracle BAM Server, 24-16
- creating a custom publish type, 24-8
- creating in Oracle JDeveloper, 24-2
- definition, 24-1
- demos, 24-3
- evaluation time, 24-4
- fault sensors, 24-2
- integration with Oracle Business Activity Monitoring, 24-15
- public views, 24-19
- publish types, 24-2
- sensor actions XSD schema file, 24-23
- sensorAction.xml file, 24-2
- sensor.xml file, 24-2
- use cases, 24-1
- variable sensors, 24-1
- viewing definitions, 24-11
- viewing metadata, 24-11
- sensor.xml file, 24-4, 24-5
- sequential list of approvers
 - configuring, 26-34
 - definition, 26-11, 26-34
 - workflow participant type, 26-11, 26-34
- service components
 - adding to an application, 2-14
 - editing, 2-16
 - overview, 2-6
- service names
 - in adapters, 11-19
- Services
 - definition, 11-10
- services
 - adding, 2-18
 - introduction, 2-3
- setting folder permissions, 32-9
- setting up, 26-56, 28-24
- simple value assert
 - overview, 43-2
- single approver
 - configuring, 26-24
 - definition, 26-10, 26-24
 - workflow participant type, 26-10, 26-24
- SMS
 - notifications support, 23-9
- SOA applications
 - design recommendations, 3-1
- SOA composite
 - creating, 2-11
- SOA composite application
 - deploying, 3-2
 - designing, 2-9
 - overview, 1-11
 - testing, 3-8
- SOA composite applications
 - troubleshooting, 3-8
- SOA Composite Editor
 - overview, 2-1
- SOA project
 - creating in Oracle JDeveloper, 2-10
- SOAP
 - security in SOAP Web services, 28-3
 - support in workflow services, 28-1
 - using with the task query service, 28-8
- SOAP headers, 12-27
 - receiving in BPEL, 12-27
 - sending in BPEL, 12-28
- SOAP interface
 - invoking a BPEL process, 20-2
- SOAP services
 - invoking a BPEL process, 20-1
 - performance issues, 18-1
 - using Java code, 18-1
- SOAP-encoded arrays
 - not supported, 12-25
- Source window, 11-7
 - definition, 11-7
- sources, message, 36-1
- Specifying Operation or Event Subscription Properties, 5-19
 - Priority, 5-19
 - Validate Syntax (XSD), 5-20
- standard faults
 - definition, 17-3
- string functions in calculations, 32-4
- strings
 - concatenating, 12-12
 - converting to an XML element, 12-25
- Structure window
 - definition, 11-11
 - location of in Oracle JDeveloper, 11-4
- style sheets
 - using for attachments, 26-59
- switch activity
 - adding, 16-3
 - in conditional branching logic, 16-2
- synchronous callbacks, 13-1
 - operational concepts, 13-5
 - required ports, 13-2
 - syncMaxWaitTime property, 13-2
- synchronous interactions
 - BPEL process as the client, 22-3
 - BPEL process as the service, 22-3
 - definition, 22-2
 - returning faults, 17-7
- synchronous processes
 - timeouts, 19-4
- synchronous reply, 6-19
- synchronous services

- callbacks with the partner link and invoke activity, 13-2
- calling, 13-5
- invoke activities, 13-4
- partnerLinkTypes, 13-3
- port types, 13-4
- ports, 13-2
- syncMaxWaitTime property
 - in synchronous callbacks, 13-2
 - increasing to prevent timeouts, 19-4

T

- task action time limits
 - specifying, 26-27, 26-31, 26-34, 26-36
- task approvers
 - specifying the number of, 26-33
- task conditions
 - abruptly completing a condition, 26-41
- task display form
 - creating, 27-5, 27-11
 - definition, 26-12, 27-1
 - deploying, 27-24, 27-26
 - displaying, 27-28
 - troubleshooting, 27-30
- .task file
 - associating with a BPEL process, 26-12, 26-69
 - definition, 26-2, 26-12, 26-13
- task flow
 - ADF, task display form for human tasks, 27-2
- task history
 - specifying in human task activities, 26-78
- task initiator
 - specifying, 26-72
- task metadata service
 - definition, 26-8
 - EJB, SOAP, and Java support, 28-2, A-5
 - supported task operations, 28-11
 - WSDL file location, 28-3
- task notification
 - editing notification messages, 26-55
 - making e-mails actionable, 26-56, 28-24
 - notifying recipients of changes to task status, 26-54
 - overview, 26-53
 - reminders, 28-28
 - securing notifications, 26-56, 28-28
 - setting up reminders, 26-55
 - task attachments with e-mail notifications, 26-57, 28-27
- task outcome
 - specifying, 26-17
- task owner
 - specifying by browsing the user directory, 26-18
 - specifying in human task activities, 26-78
 - specifying through XPath expressions, 26-20
- task parameters
 - specifying, 26-72
- task participants
 - allowing all participants to invite other

- participants, 26-40
- assigning task participants by name or expression, 26-25, 26-29, 26-32, 26-35, 26-37
- bypassing, 26-26, 26-30, 26-33, 26-36
- dynamically assigning task participants by expression, 26-26, 26-29, 26-33, 26-36, 26-38
- dynamically assigning with the assignment service, 28-32
- inviting additional task participants, 26-27, 26-34, 26-36
- sharing attachments and comments, 26-30, 26-38
- specifying, 26-22
- task payload data structure
 - specifying, 26-21
- task priority
 - specifying, 26-16, 26-72
- task query service
 - definition, 26-8
 - EJB, SOAP, and Java support, 28-2, A-5
 - supported task operations, 28-7
 - using over SOAP, 28-8
 - WSDL file location, 28-2
- task reminders
 - setting up, 26-55
- task reports service
 - EJB, SOAP, and Java support, 28-2, A-5
- task routing service
 - definition, 26-8
- task service
 - definition, 26-8
 - EJB, SOAP, and Java support, 28-2, A-5
 - supported task operations, 28-4
 - WSDL file location, 28-2
- task title
 - specifying, 26-16, 26-71
- tasks
 - definition, 26-2
 - escalating, renewing, or ending a task, 26-46
 - notifications and reminders, 28-20
 - overriding exception management, 26-60
- TCP tunneling
 - definition, 14-8
 - setting up a TCP listener for asynchronous services, 14-9
 - setting up a TCP listener for synchronous services, 14-8
- terminate activity
 - definition, 17-9
 - fault handling, 17-9
- test suites
 - components, 43-3
 - creating, 43-6
 - definition, 43-2
 - deploying from Oracle JDeveloper, 43-21
 - limitations on multibyte character names, 43-6
 - running from Oracle BPEL Control, 43-22
 - viewing test results, 43-22
- testing
 - SOA composite application, 3-8
- throw activity

- throwing internal faults, 17-6
- time
 - assigning, 12-13
- time dimensions, 32-13
- time duration format, 19-2
- time stamp field, 32-4
- time zones, changing, 29-47
- timeouts
 - designing, 19-4
 - increasing the syncMaxWaitTime property, 19-4
 - of BPEL processes, 19-1
 - use case, 19-1
 - using pick activities, 19-1
 - using the wait activity, 19-4
 - with synchronous processes, 19-4
- transform activity
 - creating, 4-1
- transformation functions
 - Component Palette, 11-9
- transformations
 - adding XSLT constructs, 4-21
 - auto mapping, 4-25
 - auto mapping with confirmation, 4-27
 - chaining functions, 4-16
 - correcting memory errors, 4-39
 - creating, 4-1
 - creating a map file from imported schemas, 4-3
 - creating a new map file, 4-2
 - creating an XSL map from an XSL stylesheet, 4-1
 - customizing sample XML generation, 4-39
 - dictionaries, 4-29
 - editing functions, 4-16
 - editing XPath expressions, 4-19
 - error when mapping duplicate elements, 4-12
 - functions, 4-15
 - functions prefixed with xp20 or orcl, 4-15
 - generating optional elements, 4-40
 - generating reports, 4-38
 - ignoring elements, 4-32
 - linking source target nodes, 4-14
 - map parameter and variable creation, 4-29
 - named templates in functions, 4-17
 - repeating elements, 4-23
 - replacing schemas, 4-32
 - rules, 4-12
 - searching source and target nodes, 4-31
 - setting constant values, 4-14
 - setting the maximum depth, 4-40
 - setting the number of repeating elements, 4-39
 - testing the map file, 4-35
 - use case, 4-1
 - using arrays, 4-23
 - using the XSLT Mapper, 4-12
 - using XQuery and XSLT, 12-4
 - viewing unmapped target nodes, 4-28
 - xsl choose conditional processing, 4-22
 - xsl if conditional processing, 4-21
- troubleshooting
 - applications, 3-8
 - task display form, 27-30

- tutorials
 - correlation sets, 14-14
 - XML facades, 18-3

U

- use cases
 - human workflow vacation request
 - example, 26-82
- user directory
 - selecting notification recipients by browsing the directory, 23-12
- user metadata service
 - definition, 26-9
 - EJB, SOAP, and Java support, 28-2, A-5
 - supported task operations, 28-12
 - WSDL file location, 28-3
- using domain value maps, 9-9
- using domain value maps a transformation, 9-11
- using error handling, 7-5
- using lookupValue functions, 9-14
- using Oracle Mediator error handling, 7-5
- using xref
 - lookupXRef Function, 10-14
 - markForDelete function, 10-18
 - populateXRefRow function, 10-9

V

- vacation rules
 - worklist, 29-28
- Validate Syntax (XSD), 5-20
- Validate Syntax (XSD) property, 5-20
- validateXML property
 - limitations on setting to true for notification services, 23-13
- validation
 - limitations on setting to true for notification services, 23-13
 - of XML data with bpelx
 - validate, 12-20
- variable sensors
 - definition, 24-1
- variables
 - complex type, 12-10
 - copying data between, 12-10
 - initializing with expression constants, 12-9
 - initializing with literal XML, 12-9
- voice mail
 - dynamically setting telephone numbers, 23-11
 - notifications support, 23-10

W

- wait activity
 - code example, 19-4
 - definition, 19-4
- Web interfaces
 - interacting with BPEL processes, 20-1
- Web Service/SOAP interface
 - invoking a BPEL process, 20-2

- while activity
 - in conditional branching logic, 16-4
- wires
 - overview, 2-9
 - using, 2-21
 - wiring a service component and reference, 2-24
- WordML style sheets
 - using for attachments, 26-59
- workflow context
 - creating on behalf of a user, 28-4
- workflow functions
 - overview, 28-1
- workflow service
 - EJB references, A-7
- workflow service clients, A-3
 - interface, A-6
- workflow services
 - abruptly completing a condition, 26-41
 - actionable e-mails, 26-56, 28-24
 - allowing all participants to invite other participants, 26-40
 - assigning task participants by name or expression, 26-25, 26-29, 26-32, 26-35, 26-37
 - assignment service configuration, 28-30
 - associating the human task activity with a BPEL process, 26-69
 - associating the human task definition with a BPEL process, 26-12
 - bypassing task participants, 26-26, 26-30, 26-33, 26-36
 - definition, 26-1
 - deprecated functions, 28-37
 - dynamically assigning task participants by expression, 26-26, 26-29, 26-33, 26-36, 26-38
 - editing notification messages, 26-55
 - EJB support, 28-1
 - escalate after policy, 26-51
 - escalating, renewing, or ending a task, 26-46
 - escalation and expiration policy overview, 26-47
 - escalation rules, 26-58
 - expire after policy, 26-49
 - external routing service task participant, 26-38
 - features, 26-2
 - functions, 28-36
 - FYI assignee task participant, 26-37
 - group vote task participant, 26-27
 - group voting details, 26-30
 - Human Task editor
 - definition, 26-2
 - identification key, 26-78
 - identity service, 26-8
 - including the task history of other tasks, 26-78
 - inviting additional task participants, 26-27, 26-34, 26-36
 - Java support, 28-1
 - management chain task participant, 26-31
 - multilingual settings, 26-59, 28-23
 - never expire policy, 26-49
 - notification
 - definition, 26-2
 - notification contents, 28-22
 - notification preferences, 26-53
 - notification service, 26-8, 28-21
 - notifications, 28-20
 - notifying recipients of changes to task status, 26-54
 - number of task approvers, 26-33
 - overriding default exception management, 26-60
 - overview, 28-1
 - renew after policy, 26-50
 - routing slip
 - definition, 26-2, 26-27
 - runtime config service, 26-9
 - scope name and global task variable name, 26-78
 - securing notifications, 26-56, 28-28
 - security model, 28-3
 - sequential list of approvers task participant, 26-34
 - setting up reminders, 26-55
 - sharing attachments and comments with task participants, 26-30, 26-38
 - single approver task participant, 26-24
 - SOAP support, 28-1
 - specifying a task initiator and task priority, 26-72
 - specifying a task title, 26-71
 - specifying callback classes, 26-61
 - specifying class names, 26-39
 - specifying task parameters, 26-72
 - style sheets in attachments, 26-59
 - support for identity service, 28-8
 - support for JSP-based forms, 26-2
 - support for task delegation, escalation, and reapproval, 26-2
 - support for task expiration and automatic renewal, 26-2
 - task
 - definition, 26-2
 - task assignment and routing
 - definition, 26-2
 - task attachments with e-mail notifications, 26-57, 28-27
 - task display form, 26-12, 27-1
 - .task file
 - definition, 26-2, 26-12, 26-13
 - task metadata service, 26-8
 - task notifications, 28-20
 - task outcome, 26-17
 - task owner, 26-78
 - task owner specification through the user directory, 26-18
 - task owner specification through XPath expressions, 26-20
 - task participants, 26-22
 - task payload data structure, 26-21
 - task priority, 26-16
 - task query service, 26-8
 - task routing and customization in BPEL callbacks, 26-62, 26-79
 - task routing service, 26-8
 - task service, 26-8

- task title, 26-16
- time limits for acting on tasks, 26-27, 26-31, 26-34, 26-36
- user metadata service, 26-9
- viewing BPEL callbacks, 26-75
- WordML style sheets in attachments, 26-59
- worklist
 - definition, 26-2
- XPath extension functions, 28-36
- worklist
 - administration functions, 29-33
 - changing the display, 29-35
 - logging in, 29-2
 - mapping flex fields, 29-36
 - reports, 29-40, 29-41
 - rules, 29-28
 - Task Details page, acting on tasks, 29-15
 - Task Listing page contents, 29-4
 - Task Listing page, customizing, 29-5
 - vacation rules, 29-28
- worklist clients
 - building for workflow services, A-1
 - classpaths for clients using local EJBs, A-7
 - classpaths for clients using remote EJBs, A-7
 - classpaths for clients using SOAP, A-6
 - customizing, A-1
 - packages and classes for, A-3
- worklists
 - definition, 26-2
- WS-Addressing, 14-6
 - definition, 14-6, 14-7
 - sending correlation IDs, 14-6
- WSDL files
 - definition, 11-5
 - differences between document-literal styles and RPC styles, 12-2, 12-26
 - location for identity service, 28-2
 - location for notification service, 28-2
 - location for runtime config service, 28-3
 - location for task metadata service, 28-3
 - location for task query service, 28-2
 - location for task service, 28-2
 - location for user metadata service, 28-3
 - modifying to generate a fault, 17-5
 - specifying when creating a partner link, 11-14

X

- XML assert
 - overview, 43-3
- XML data in BPEL, 12-2
- XML data manipulation
 - bpelx:append extension, 12-15
 - bpelx:copyList extension, 12-19
 - bpelx:insertAfter extension, 12-16
 - bpelx:insertBefore extension, 12-15
 - bpelx:remove extension, 12-17
 - bpelx:rename extension, 12-18
 - bpelx:validate extension, 12-20
- XML documents
 - manipulating, 12-2, 12-4
 - overview, 12-2, 12-4
- XML facades
 - definition, 18-3
 - generating with schemac, 18-3
 - Java embedding, 18-3
 - tutorials, 18-3
- XML schema files
 - error handling, 7-5
 - fault-bindings.xml, 7-9
 - fault-policies.xml, 7-5
- XML schemas
 - message types and variable types, 12-1
- XPath, 12-2
- XPath expressions
 - assigning numeric values, 12-11
 - Boolean expressions in switch activities, 16-4
 - dynamically creating another XPath expression, 12-22
 - dynamically setting e-mail addresses and telephone numbers, 23-11
 - editing in transformations, 4-19
 - examples, 12-3
 - fetching a data sequence element, 12-22
 - in conditional branching logic, 16-1
 - specifying a task owner, 26-20
- xpath function
 - dvm
 - lookupValue function, 9-9
 - lookupValue1M function, 9-10
- XPath functions
 - examples, 12-3
 - in transformations, 4-15
 - indexing methods, 12-22
 - mathematical calculations, 12-11
- XPath queries
 - copying data, 12-10
 - examples, 12-3
- XQuery, 12-2, 12-4
- xref
 - lookupXRef function, 10-13
 - exception reasons, 10-14
 - parameters, 10-14
 - using, 10-14
 - lookupXRef1M function, 10-16
 - exception reasons, 10-17
 - parameters, 10-17
 - markForDelete function, 10-17
 - exception reasons, 10-18
 - parameters, 10-17
 - using, 10-18
 - populateXRefRow function, 10-7
 - modes, 10-7
 - parameters, 10-7
 - using, 10-9
 - populateXRefRow1M function, 10-12
 - modes, 10-12
 - parameters, 10-12
- XSD file
 - cross references, 10-20

- xsl choose
 - conditional processing, 4-22
- xsl if
 - conditional processing, 4-21
- XSL map
 - creating from an XSL stylesheet, 4-1
- XSL processing, 36-5
- XSL stylesheet
 - creating an XSL map, 4-1
- XSLT, 12-2, 12-4
- XSLT constructs
 - adding in transformations, 4-21
- XSLT Mapper
 - adding XSLT constructs, 4-21
 - auto mapping, 4-25
 - auto mapping with confirmation, 4-27
 - chaining functions, 4-16
 - correcting memory errors when generating reports, 4-39
 - creating a map file, 4-8
 - creating a map file from imported schemas, 4-3
 - creating a new map file, 4-2
 - creating a transform activity, 4-1
 - creating an XSL map from an XSL stylesheet, 4-1
 - customizing sample XML generation for transformations, 4-39
 - dictionaries, 4-29
 - editing functions, 4-16
 - editing XPath expressions, 4-19
 - error when mapping duplicate elements, 4-12
 - functions, 4-15
 - functions prefixed with xp20 or orcl, 4-15
 - generating optional elements, 4-40
 - generating reports, 4-38
 - ignoring elements, 4-32
 - layout in Oracle JDeveloper, 4-8
 - linking source and target nodes, 4-14
 - map parameter and variable creation, 4-29
 - named templates in functions, 4-17
 - repeating elements, 4-23
 - replacing schemas, 4-32
 - rules, 4-12
 - searching source and target nodes, 4-31
 - setting constant values, 4-14
 - setting the maximum depth, 4-40
 - setting the number of repeating elements, 4-39
 - testing the map file, 4-35
 - use case, 4-1
 - using, 4-12
 - using arrays, 4-23
 - viewing unmapped target nodes, 4-28
 - xsl choose conditional processing, 4-22
 - xsl if conditional processing, 4-21

