

Oracle® Fusion Middleware

Developer's Guide for Oracle SOA Suite, Volume 1

11g Release 1 (11.1.1.5.0 Feature Pack)

E10224-10

September 2011

Copyright © 2005, 2011, Oracle and/or its affiliates. All rights reserved.

Primary Author: Virginia Beecher, Anirban Ghosh, Mark Kennedy, Alex Prazma, Richard Smith, and Carol Thom

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

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

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

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

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

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

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

Contents

Preface	lv
Audience	lv
Documentation Accessibility	lv
Related Documents	lvi
Conventions	lvi

Part I Introduction to Oracle SOA Suite

1 Introduction to Building Applications with Oracle SOA Suite

1.1	Introduction to Service-Oriented Architecture.....	1-1
1.2	Introduction to Services	1-1
1.3	Introduction to Oracle SOA Suite.....	1-2
1.4	Standards Used by Oracle SOA Suite to Enable SOA	1-2
1.5	Service Component Architecture within SOA Composite Applications.....	1-3
1.5.1	Service Components.....	1-4
1.5.2	Binding Components	1-5
1.5.3	Wires.....	1-6
1.6	Runtime Behavior of a SOA Composite Application	1-6
1.6.1	Service Infrastructure	1-7
1.6.2	Service Engines	1-8
1.6.3	Deployed Service Archives	1-8
1.7	Approaches for Designing SOA Composite Applications.....	1-8
1.8	Learning Oracle SOA Suite.....	1-8

2 Developing SOA Composite Applications with Oracle SOA Suite

2.1	Creating a SOA Application.....	2-1
2.1.1	How to Create a SOA Application and Project	2-1
2.1.2	What Happens When You Create a SOA Application and Project.....	2-3
2.1.3	What You May Need to Know About Opening the composite.xml File Through a SOA-MDS Connection 2-6	
2.2	Adding Service Components	2-6
2.2.1	How to Add a Service Component	2-6
2.2.2	What You May Need to Know About Adding and Deleting a Service Component .	2-8
2.2.3	How to Edit a Service Component.....	2-9
2.3	Adding Service Binding Components	2-10

2.3.1	How to Add a Service Binding Component	2-10
2.3.2	How to Add a WSDL for a Web Service	2-12
2.3.3	How to View Schemas	2-15
2.3.4	How to Edit a Service Binding Component.....	2-16
2.3.5	What You May Need to Know About Adding and Deleting Services	2-16
2.4	Adding Reference Binding Components.....	2-16
2.4.1	How to Add a Reference Binding Component	2-16
2.4.2	What You May Need to Know About Adding and Deleting References.....	2-18
2.4.3	What You May Need to Know About WSDL References.....	2-19
2.4.4	What You May Need to Know About Mixed Message Types in a WSDL File	2-19
2.4.5	What You May Need to Know About Invoking the Default Revision of a Composite	2-19
2.5	Adding Wires	2-20
2.5.1	How to Wire a Service and a Service Component	2-20
2.5.2	How to Wire a Service Component and a Reference	2-21
2.5.3	What You May Need to Know About Adding and Deleting Wires	2-23
2.6	Adding Security	2-24
2.7	Deploying a SOA Composite Application	2-25
2.7.1	How to Invoke Deployed Composites	2-25
2.8	Managing and Testing a SOA Composite Application.....	2-25
2.8.1	How to Manage Deployed Composites	2-26
2.8.2	How to Test a Deployed Composite	2-29

3 Introduction to the SOA Sample Application

3.1	Introduction to the Fusion Order Demo.....	3-1
3.1.1	Store Front Module.....	3-1
3.1.2	WebLogic Fusion Order Demo Application	3-2
3.2	Setting Up the Fusion Order Demo Application.....	3-3
3.2.1	Task 1: Install Oracle JDeveloper Studio.....	3-3
3.2.2	Task 2: Install the Fusion Order Demo Application	3-3
3.2.3	Task 3: Install Oracle SOA Suite	3-4
3.3	Taking a Look at the WebLogic Fusion Order Demo Application.....	3-6
3.3.1	Project Applications of the WebLogic Fusion Order Demo Application	3-7
3.3.2	The composite.xml File	3-7
3.4	Understanding the OrderBookingComposite Flow.....	3-8
3.5	Deploying Fusion Order Demo	3-12
3.5.1	Task 1: Create a Connection to an Oracle WebLogic Server	3-12
3.5.2	(Optional) Task 2: Create a Connection to the Oracle BAM Server	3-14
3.5.3	Task 3: Install the Schema for the Fusion Order Demo Application.....	3-15
3.5.4	Task 4: Set the Configuration Property for the Store Front Module	3-16
3.5.5	Task 5: Edit the Database Connection	3-18
3.5.6	Task 6: Deploy the Store Front Module.....	3-19
3.5.7	Task 7: Deploy the WebLogic Fusion Order Demo Application	3-20
3.6	Running Fusion Order Demo.....	3-23
3.7	Viewing Data Sent to Oracle BAM Server	3-24
3.8	Undeploying the Composites for the WebLogic Fusion Order Demo Application	3-24

Part II Using the BPEL Process Service Component

4 Getting Started with Oracle BPEL Process Manager

4.1	Introduction to the BPEL Process Service Component	4-1
4.1.1	How to Add a BPEL Process Service Component	4-1
4.2	Introduction to Activities.....	4-6
4.3	Introduction to Partner Links.....	4-7
4.4	Creating a Partner Link	4-9
4.4.1	How to Create a Partner Link	4-9
4.4.1.1	Partner Links for an Outbound Adapter	4-10
4.4.1.2	Partner Links for an Inbound Adapter.....	4-11
4.4.1.3	Partner Links from an Abstract WSDL to Call a Service	4-11
4.4.1.4	Partner Links from an Abstract WSDL to Implement a Service.....	4-11
4.4.1.5	Partner Links and Human Tasks or Business Rules.....	4-12
4.4.1.6	Partner Links from an Existing Human Task, Business Rule, or Oracle Mediator.....	4-12
4.5	Introduction to Technology Adapters	4-13
4.6	Introduction to BPEL Process Monitors	4-14

5 Introduction to Interaction Patterns in a BPEL Process

5.1	Introduction to One-Way Messages.....	5-1
5.2	Introduction to Synchronous Interactions.....	5-2
5.3	Introduction to Asynchronous Interactions.....	5-3
5.4	Introduction to Asynchronous Interactions with a Timeout.....	5-4
5.5	Introduction to Asynchronous Interactions with a Notification Timer.....	5-5
5.6	Introduction to One Request, Multiple Responses	5-6
5.7	Introduction to One Request, One of Two Possible Responses	5-7
5.8	Introduction to One Request, a Mandatory Response, and an Optional Response.....	5-8
5.9	Introduction to Partial Processing	5-9
5.10	Introduction to Multiple Application Interactions	5-10

6 Manipulating XML Data in a BPEL Process

6.1	Introduction to Manipulating XML Data in BPEL Processes.....	6-2
6.1.1	XML Data in BPEL.....	6-2
6.1.2	Data Manipulation and XPath Standards	6-2
6.2	Delegating XML Data Operations to Data Provider Services	6-5
6.2.1	How to Create an Entity Variable	6-7
6.2.1.1	Understanding How SDO Works in the Inbound Direction.....	6-7
6.2.1.2	Understanding How SDO Works in the Outbound Direction	6-8
6.2.1.3	Creating an Entity Variable and Choosing a Partner Link.....	6-8
6.2.1.4	Creating a Binding Key.....	6-9
6.3	Using Standalone SDO-based Variables.....	6-11
6.3.1	How to Declare SDO-based Variables.....	6-11
6.3.2	How to Convert from XML to SDO	6-12
6.4	Initializing a Variable with Expression Constants or Literal XML.....	6-13
6.4.1	How To Assign a Literal XML Element	6-13

6.5	Copying Between Variables	6-14
6.5.1	How to Copy Between Variables.....	6-14
6.5.2	Initializing Variables with an Inline from-spec in BPEL 2.0.....	6-15
6.6	Accessing Fields in Element and Message Type Variables	6-15
6.6.1	How to Access Fields Within Element-Based and Message Type-Based Variables	6-15
6.7	Assigning Numeric Values.....	6-17
6.7.1	How to Assign Numeric Values.....	6-17
6.8	Using Mathematical Calculations with XPath Standards	6-17
6.8.1	How To Use Mathematical Calculations with XPath Standards	6-17
6.9	Assigning String Literals.....	6-18
6.9.1	How to Assign String Literals.....	6-18
6.10	Concatenating Strings	6-18
6.10.1	How to Concatenate Strings.....	6-19
6.11	Assigning Boolean Values	6-19
6.11.1	How to Assign Boolean Values	6-19
6.12	Assigning a Date or Time	6-20
6.12.1	How to Assign a Date or Time.....	6-20
6.13	Manipulating Attributes	6-21
6.13.1	How to Manipulate Attributes	6-21
6.14	Manipulating XML Data with bpelx Extensions	6-22
6.14.1	How to Use bpelx:append	6-23
6.14.1.1	bpelx:append in BPEL 1.1.....	6-23
6.14.1.2	bpelx:append in BPEL 2.0.....	6-24
6.14.2	How to Use bpelx:insertBefore	6-24
6.14.2.1	bpelx:insertBefore in BPEL 1.1.....	6-24
6.14.2.2	bpelx:insertBefore in BPEL 2.0.....	6-25
6.14.3	How to Use bpelx:insertAfter	6-26
6.14.3.1	bpelx:insertAfter in BPEL 1.1.....	6-26
6.14.3.2	bpelx:insertAfter in BPEL 2.0.....	6-27
6.14.4	How to Use bpelx:remove	6-27
6.14.4.1	bpelx:remove in BPEL 1.1.....	6-28
6.14.4.2	bpelx:remove in BPEL 2.0.....	6-29
6.14.5	How to Use bpelx:rename and XSD Type Casting	6-29
6.14.5.1	bpelx:rename in BPEL 1.1.....	6-29
6.14.5.2	bpelx:rename in BPEL 2.0.....	6-31
6.14.6	How to Use bpelx:copyList	6-31
6.14.6.1	bpelx:copyList in BPEL 1.1.....	6-32
6.14.6.2	bpelx:copyList in BPEL 2.0.....	6-33
6.14.7	How to Use Assign Extension Attributes.....	6-34
6.14.7.1	ignoreMissingFromData Attribute	6-34
6.14.7.2	insertMissingToData Attribute.....	6-34
6.14.7.3	keepSrcElementName Attribute	6-35
6.15	Validating XML Data	6-35
6.15.1	How to Validate XML Data in BPEL 1.1.....	6-35
6.15.2	How to Validate XML Data in BPEL 2.0.....	6-35
6.16	Using Element Variables in Message Exchange Activities in BPEL 2.0.....	6-36
6.17	Mapping WSDL Message Parts in BPEL 2.0	6-37

6.17.1	How to Map WSDL Message Parts.....	6-38
6.17.2	What Happens When You Map WSDL Message Parts.....	6-39
6.18	Importing Process Definitions in BPEL 2.0	6-44
6.19	Manipulating XML Data Sequences That Resemble Arrays	6-45
6.19.1	How to Statically Index into an XML Data Sequence That Uses Arrays.....	6-45
6.19.2	How to Use SOAP-Encoded Arrays	6-46
6.19.2.1	SOAP-Encoded Arrays in BPEL 2.0	6-47
6.19.3	How to Determine Sequence Size	6-47
6.19.4	How to Dynamically Index by Applying a Trailing XPath to an Expression.....	6-48
6.19.4.1	Applying a Trailing XPath to the Result of getVariableData	6-48
6.19.4.2	Using the bpelx:append Extension to Append New Items to a Sequence.....	6-49
6.19.4.3	Merging Data Sequences	6-49
6.19.4.4	Generating Functionality Equivalent to an Array of an Empty Element.....	6-50
6.19.5	What You May Need to Know About Using the Array Identifier	6-51
6.20	Converting from a String to an XML Element.....	6-51
6.20.1	How To Convert from a String to an XML Element.....	6-51
6.21	Understanding Document-Style and RPC-Style WSDL Differences.....	6-52
6.21.1	How To Use RPC-Style Files	6-52
6.22	Manipulating SOAP Headers in BPEL	6-53
6.22.1	How to Receive SOAP Headers in BPEL	6-53
6.22.2	How to Send SOAP Headers in BPEL	6-54
6.23	Declaring Extension Namespaces in BPEL 2.0	6-55
6.23.1	How to Declare Extension Namespaces	6-55
6.23.2	What Happens When You Create an Extension	6-55

7 Invoking a Synchronous Web Service from a BPEL Process

7.1	Introduction to Invoking a Synchronous Web Service.....	7-1
7.2	Invoking a Synchronous Web Service	7-2
7.2.1	How to Invoke a Synchronous Web Service.....	7-2
7.2.2	What Happens When You Invoke a Synchronous Web Service	7-4
7.2.2.1	Partner Link in the BPEL Code.....	7-4
7.2.2.2	Partner Link Type and Port Type in the BPEL Code	7-4
7.2.2.3	Invoke Activity for Performing a Request.....	7-5
7.2.2.4	Synchronous Invocation in BPEL Code	7-5
7.3	Specifying Timeout Values.....	7-6
7.3.1	How To Specify Timeout Values.....	7-6
7.3.2	What You May Need to Know About SyncMaxWaitTime and Synchronous Requests Not Timing Out	7-6
7.4	Calling a One-Way Mediator with a Synchronous BPEL Process	7-7

8 Invoking an Asynchronous Web Service from a BPEL Process

8.1	Introduction to Invoking an Asynchronous Web Service.....	8-1
8.2	Invoking an Asynchronous Web Service	8-2
8.2.1	How to Invoke an Asynchronous Web Service.....	8-2
8.2.1.1	Adding a Partner Link for an Asynchronous Service	8-2
8.2.1.2	Adding an Invoke Activity	8-3

8.2.1.3	Adding a Receive Activity	8-4
8.2.1.4	Performing Additional Activities.....	8-5
8.2.2	What Happens When You Invoke an Asynchronous Web Service	8-6
8.2.2.1	portType Section of the WSDL File.....	8-6
8.2.2.2	partnerLinkType Section of the WSDL File.....	8-6
8.2.2.3	Partner Links Section in the BPEL File	8-7
8.2.2.4	Composite Application File	8-7
8.2.2.5	Invoke and Receive Activities.....	8-8
8.2.2.6	createInstance Attribute for Starting a New Instance	8-8
8.2.2.7	Dehydration Points for Maintaining Long-Running Asynchronous Processes..	8-9
8.2.2.8	Multiple Runtime Endpoint Locations.....	8-9
8.2.3	What You May Need to Know About Limitations on BPEL 2.0 IMA Support	8-9
8.2.4	What Happens When You Specify a Conversation ID.....	8-10
8.2.4.1	bpelx:conversationId in BPEL 1.1.....	8-10
8.2.4.2	bpelx:conversationId in BPEL 2.0.....	8-10
8.3	Using a Dynamic Partner Link at Runtime	8-11
8.3.1	How To Add and Use a Dynamic Partner Link at Runtime	8-11
8.4	Using WS-Addressing in an Asynchronous Service.....	8-12
8.4.1	How to Use WS-Addressing in an Asynchronous Service.....	8-13
8.4.1.1	Using TCP Tunneling to See Messages Exchanged Between Programs	8-13
8.5	Using Correlation Sets in an Asynchronous Service	8-15
8.5.1	How to Use Correlation Sets in an Asynchronous Service.....	8-15
8.5.1.1	Step 1: Creating a Project.....	8-15
8.5.1.2	Step 2: Configuring Partner Links and File Adapter Services	8-16
8.5.1.3	Step 3: Creating Three Receive Activities	8-20
8.5.1.4	Step 4: Creating Correlation Sets.....	8-21
8.5.1.5	Step 5: Associating Correlation Sets with Receive Activities.....	8-22
8.5.1.6	Step 6: Creating Property Aliases.....	8-23
8.5.1.7	Step 7: Reviewing WSDL File Content	8-25
8.5.2	What You May Need to Know About Setting Correlations for an IMA Using a fromParts Element With Multiple Parts	8-26

9 Using Parallel Flow in a BPEL Process

9.1	Introduction to Parallel Flows in BPEL Processes.....	9-1
9.2	Creating a Parallel Flow	9-2
9.2.1	How to Create a Parallel Flow	9-2
9.2.2	What Happens When You Create a Parallel Flow	9-3
9.2.3	Synchronizing the Execution of Activities in a Flow Activity	9-5
9.2.4	How to Create Synchronization Between Activities Within a Flow Activity	9-6
9.2.5	What Happens When You Create Synchronization Between Activities Within a Flow Activity	9-9
9.2.6	What You May Need to Know About Join Conditions in Target Activities.....	9-11
9.3	Customizing the Number of Parallel Branches	9-11
9.3.1	Customizing the Number of Flow Activities with the flowN Activity in BPEL 1.1	9-11
9.3.1.1	How to Create a flowN Activity.....	9-13
9.3.1.2	What Happens When You Create a FlowN Activity	9-14
9.3.2	Processing Multiple Sets of Activities with the forEach Activity in BPEL 2.0.....	9-16

9.3.2.1	How to Create a forEach Activity	9-18
9.3.2.2	What Happens When You Create a forEach Activity	9-20

10 Using Conditional Branching in a BPEL Process

10.1	Introduction to Conditional Branching	10-1
10.2	Defining Conditional Branching.....	10-2
10.2.1	Defining Conditional Branching with the Switch Activity in BPEL 1.1	10-2
10.2.1.1	How to Create a Switch Activity	10-3
10.2.1.2	What Happens When You Create a Switch Activity.....	10-4
10.2.2	Defining Conditional Branching with the If Activity in BPEL 2.0.....	10-5
10.2.2.1	How to Create an If Activity	10-6
10.2.2.2	What Happens When You Create an If Activity.....	10-7
10.3	Creating a While Activity to Define Conditional Branching.....	10-8
10.3.1	How To Create a While Activity	10-8
10.3.2	What Happens When You Create a While Activity	10-9
10.4	Creating a repeatUntil Activity to Define Conditional Branching	10-10
10.4.1	How to Create a repeatUntil Activity	10-10
10.4.2	What Happens When You Create a repeatUntil Activity.....	10-11
10.5	Specifying XPath Expressions to Bypass Activity Execution.....	10-12
10.5.1	How to Specify XPath Expressions to Bypass Activity Execution	10-13
10.5.2	What Happens When You Specify XPath Expressions to Bypass Activity Execution	10-14

11 Using Fault Handling in a BPEL Process

11.1	Introduction to a Fault Handler.....	11-1
11.2	Introduction to BPEL Standard Faults.....	11-3
11.2.1	BPEL 1.1 Standard Faults.....	11-3
11.2.2	BPEL 2.0 Standard Faults.....	11-3
11.2.2.1	Fault Handling Order of Precedence in BPEL 2.0.....	11-4
11.3	Introduction to Categories of BPEL Faults.....	11-5
11.3.1	Business Faults	11-5
11.3.2	Runtime Faults	11-5
11.3.2.1	bindingFault	11-6
11.3.2.2	remoteFault.....	11-6
11.3.2.3	replayFault.....	11-6
11.4	Using the Fault Management Framework	11-6
11.4.1	How to Design a Fault Policy	11-7
11.4.1.1	Understanding How Fault Policy Binding Resolution Works.....	11-7
11.4.1.2	Creating a Fault Policy File for Automated Fault Recovery	11-8
11.4.1.3	Associating a Fault Policy with Fault Policy Binding	11-12
11.4.1.4	Additional Fault Policy and Fault Policy Binding File Samples.....	11-13
11.4.1.5	Designing a Fault Policy with Multiple Rejection Handlers.....	11-16
11.4.2	How to Execute a Fault Policy	11-17
11.4.3	How to Use a Java Action Fault Policy.....	11-17
11.4.4	What You May Need to Know About Fault Management Behavior When the Number of Instance Retries is Exceeded	11-21

11.4.5	What You May Need to Know Executing the Retry Action with Multiple Faults in the Same Flow	11-22
11.4.6	What You May Need to Know About Binding Level Retry Execution Within Fault Policy Retries	11-22
11.4.7	What You May Need to Know About Defining the ora-java Option	11-23
11.5	Catching BPEL Runtime Faults	11-24
11.5.1	How to Catch BPEL Runtime Faults.....	11-24
11.6	Getting Fault Details with the getFaultAsString XPath Extension Function	11-25
11.6.1	How to Get Fault Details with the getFaultAsString XPath Extension Function..	11-25
11.7	Throwing Internal Faults	11-25
11.7.1	How to Create a Throw Activity	11-25
11.7.2	What Happens When You Create a Throw Activity	11-26
11.8	Rethrowing Faults with the Rethrow Activity	11-26
11.8.1	How to Create a Rethrow Activity	11-26
11.8.2	What Happens When You Rethrow Faults.....	11-27
11.9	Returning External Faults	11-28
11.9.1	How to Return a Fault in a Synchronous Interaction.....	11-28
11.9.2	How to Return a Fault in an Asynchronous Interaction.....	11-28
11.10	Using a Scope Activity to Manage a Group of Activities.....	11-29
11.10.1	How to Create a Scope Activity.....	11-29
11.10.2	How to Add Descriptive Notes and Images to a Scope Activity.....	11-30
11.10.3	What Happens After You Create a Scope Activity	11-31
11.10.4	What You May Need to Know About Scopes	11-33
11.10.5	How to Use a Fault Handler Within a Scope.....	11-33
11.10.6	How to Create a Catch Activity in a Scope	11-34
11.10.7	What Happens When You Create a Catch Activity in a Scope.....	11-35
11.10.8	How to Create an Empty Activity to Insert No-Op Instructions into a Business Process	11-36
11.10.9	What Happens When You Create an Empty Activity.....	11-37
11.11	Re-executing Activities in a Scope Activity with the Replay Activity	11-37
11.11.1	How to Create a Replay Activity	11-37
11.11.2	What Happens When You Create a Replay Activity	11-38
11.12	Using Compensation After Undoing a Series of Operations	11-39
11.12.1	Using a Compensate Activity	11-39
11.12.2	How to Create a Compensate Activity	11-40
11.12.3	What Happens When You Create a compensate Activity.....	11-41
11.12.4	Using a compensateScope Activity in BPEL 2.0.....	11-41
11.12.5	How to Create a compensateScope Activity	11-41
11.12.6	What Happens When You Create a compensateScope Activity	11-42
11.13	Stopping a Business Process Instance	11-42
11.13.1	Stopping a Business Process Instance with the Terminate Activity in BPEL 1.1 ..	11-43
11.13.1.1	How to Create a Terminate Activity.....	11-43
11.13.1.2	What Happens When You Create a Terminate Activity	11-43
11.13.2	Immediately Ending a Business Process Instance with the Exit Activity in BPEL 2.0	11-43
11.13.2.1	How to Create an Exit Activity.....	11-44
11.13.2.2	What Happens When You Create an Exit Activity	11-44
11.14	Throwing Faults with Assertion Conditions	11-45

11.14.1	bpelx:postAssert and bpelx:preAssert Extensions.....	11-46
11.14.2	Use of faultName and message Attributes	11-47
11.14.3	Multiple Assertions	11-47
11.14.4	Use of Built-in and Custom XPath Functions and \$variable References	11-48
11.14.5	Assertion Condition Evaluation Logging of Events to the Instance Audit Trail ..	11-49
11.14.6	Expressions Not Evaluating to an XML Schema Boolean Type Throw a Fault ...	11-49
11.14.7	Assertion Conditions in a Standalone Assert Activity	11-49
11.14.8	How to Create Assertion Conditions.....	11-49
11.14.9	How to Disable Assertions	11-52
11.14.10	What Happens When You Create Assertion Conditions.....	11-52
12	Transaction and Fault Propagation Semantics in BPEL Processes	
12.1	Introduction to Transaction Semantics	12-1
12.1.1	Oracle BPEL Process Manager Transaction Semantics	12-1
12.1.1.1	BPELCaller Calls BPELCallee That Has bpel.config.transaction Set to requiresNew 12-2	
12.1.1.2	BPELCaller Calls BPELCallee That Has bpel.config.transaction Set to required..... 12-3	
12.2	Introduction to Execution of One-way Invocations.....	12-4
13	Incorporating Java and Java EE Code in a BPEL Process	
13.1	Introduction to Java and Java EE Code in BPEL Processes	13-1
13.2	Incorporating Java and Java EE Code in BPEL Processes.....	13-1
13.2.1	How to Wrap Java Code as a SOAP Service.....	13-1
13.2.2	What You May Need to Know About Wrapping Java Code as a SOAP Service	13-2
13.2.3	How to Embed Java Code Snippets into a BPEL Process with the bpelx:exec Tag	13-2
13.2.4	How to Embed Java Code Snippets in a BPEL Process in BPEL 2.0	13-3
13.2.5	How to Use an XML Facade to Simplify DOM Manipulation.....	13-4
13.2.6	How to Use bpelx:exec Built-in Methods	13-4
13.2.7	How to Use Java Code Wrapped in a Service Interface.....	13-5
13.3	Adding Custom Classes and JAR Files.....	13-6
13.3.1	How to Add Custom Classes and JAR Files	13-6
13.4	Using Java Embedding in a BPEL Process in Oracle JDeveloper	13-7
13.4.1	How To Use Java Embedding in a BPEL Process in Oracle JDeveloper	13-7
13.4.2	What You May Need to Know About Using thread.sleep() in a Java Embedding Activity 13-8	
13.5	Embedding Service Data Objects with bpelx:exec	13-8
13.6	Sharing a Custom Implementation of a Class with Oracle BPEL Process Manager	13-9
13.6.1	How to Configure the BPEL Connection Manager Class to Take Precedence	13-10
14	Using Events and Timeouts in BPEL Processes	
14.1	Introduction to Event and Timeout Concepts	14-1
14.2	Creating a Pick Activity to Select Between Continuing a Process or Waiting	14-2
14.2.1	How To Create a Pick Activity	14-3
14.2.2	What Happens When You Create a Pick Activity	14-5

14.2.3	What You May Need to Know About Simultaneous onMessage Branches in BPEL 2.0..	14-6
14.3	Setting Timeouts for Request-Response Operations in Receive Activities	14-7
14.3.1	Timeout Settings Relative from When the Activity is Invoked	14-7
14.3.2	Timeout Settings as an Absolute Date Time	14-8
14.3.3	Timeout Settings Computed Dynamically with an XPath Expression	14-9
14.3.4	bpelx:timeout Fault Thrown During an Activity Timeout	14-9
14.3.5	Event Added to the BPEL Instance Audit Trail During an Activity Timeout	14-10
14.3.6	Recoverable Timeout Activities During a Server Restart (Refresh Expiration Alarm Table)	14-10
14.3.7	How to Set Timeouts for Request-Response Operations in Receive Activities	14-10
14.3.8	What Happens When You Set Timeouts for Request-Response Operations in Receive Activities	14-11
14.4	Creating a Wait Activity to Set an Expiration Time	14-12
14.4.1	How To Specify the Minimum Wait Time	14-12
14.4.2	How to Create a Wait Activity	14-12
14.4.3	What Happens When You Create a Wait Activity	14-13
14.5	Specifying Events to Wait for Message Arrival with an OnEvent Branch in BPEL 2.0	14-13
14.5.1	How to Create an onEvent Branch in a Scope Activity	14-14
14.5.2	What Happens When You Create an OnEvent Branch	14-15
14.6	Setting Timeouts for Synchronous Processes	14-15

15 Coordinating Master and Detail Processes

15.1	Introduction to Master and Detail Process Coordinations	15-1
15.1.1	BPEL File Definition for the Master Process	15-3
15.1.1.1	Correlating a Master Process with Multiple Detail Processes	15-5
15.1.2	BPEL File Definition for Detail Processes	15-6
15.2	Defining Master and Detail Process Coordination in Oracle JDeveloper	15-7
15.2.1	How to Create a Master Process	15-7
15.2.2	How to Create a Detail Process	15-9
15.2.3	How to Create an Invoke Activity	15-11

16 Using the Notification Service

16.1	Introduction to the Notification Service	16-1
16.2	Introduction to Notification Channel Setup	16-3
16.3	Selecting Notification Channels During BPEL Process Design	16-3
16.3.1	How To Configure the Email Notification Channel	16-4
16.3.1.1	Setting Email Attachments	16-7
16.3.1.2	Formatting the Body of an Email Message as HTML	16-8
16.3.1.3	Using Dynamic HTML for Message Content Requires a CDATA Function	16-9
16.3.2	How to Configure the IM Notification Channel	16-9
16.3.3	How to Configure the SMS Notification Channel	16-10
16.3.4	How to Configure the Voice Notification Channel	16-12
16.3.5	How to Select Email Addresses and Telephone Numbers Dynamically	16-12
16.3.6	How to Select Notification Recipients by Browsing the User Directory	16-13
16.4	Allowing the End User to Select Notification Channels	16-14
16.4.1	How to Allow the End User to Select Notification Channels	16-14

16.4.1.1	How to Create and Send Headers for Notifications.....	16-15

17 Using Oracle BPEL Process Manager Sensors

17.1	Introduction to Sensors	17-1
17.2	Configuring Sensors and Sensor Actions in Oracle JDeveloper	17-3
17.2.1	How to Access Sensors and Sensor Actions	17-3
17.2.2	How to Configure Sensors	17-4
17.2.3	How to Configure Sensor Actions.....	17-8
17.2.4	How to Publish to Remote Topics and Queues.....	17-11
17.2.5	How to Create a Custom Data Publisher	17-12
17.2.6	How to Register the Sensors and Sensor Actions in composite.xml.....	17-14
17.3	Viewing Sensors and Sensor Action Definitions in Oracle Enterprise Manager Fusion Middleware Control	17-15

Part III Using the Oracle Mediator Service Component

18 Getting Started with Oracle Mediator

18.1	Introduction to Oracle Mediator.....	18-1
18.2	Introduction to the Mediator Editor Environment	18-3
18.3	Creating an Oracle Mediator	18-6
18.3.1	How to Create an Oracle Mediator	18-6
18.4	Configuring the Oracle Mediator Interface Definition.....	18-9
18.4.1	Creating an Oracle Mediator Without an Interface Definition	18-9
18.4.1.1	How to Create an Oracle Mediator Without an Interface Definition	18-10
18.4.1.2	What Happens When You Create an Oracle Mediator Without an Interface Definition	18-10
18.4.1.3	How to Define an Interface for an Oracle Mediator.....	18-11
18.4.2	Creating an Oracle Mediator Based on a WSDL File.....	18-13
18.4.2.1	How to Create an Oracle Mediator Based on a WSDL File.....	18-13
18.4.2.2	What Happens When You Create an Oracle Mediator from a WSDL File	18-14
18.4.3	Creating an Oracle Mediator With a One-Way Interface Definition	18-15
18.4.3.1	How to Create an Oracle Mediator with a One-Way Interface Definition	18-15
18.4.3.2	What Happens When You Create an Oracle Mediator with a One-Way Interface Definition	18-16
18.4.4	Creating an Oracle Mediator with a Synchronous Interface Definition.....	18-16
18.4.4.1	How to Create an Oracle Mediator with a Synchronous Interface Definition	18-17
18.4.4.2	What Happens When You Create an Oracle Mediator with a Synchronous Interface Definition	18-17
18.4.5	Creating an Oracle Mediator with an Asynchronous Interface Definition.....	18-18
18.4.5.1	How to Create an Oracle Mediator with an Asynchronous Interface Definition	18-18
18.4.5.2	What Happens When You Create an Oracle Mediator with an Asynchronous Interface Definition	18-19
18.4.6	Creating an Oracle Mediator for an Event Subscription.....	18-20
18.4.6.1	How to Create an Oracle Mediator for an Event Subscription.....	18-20
18.4.6.2	What Happens When You Create an Oracle Mediator for an Event Subscription.....	18-22

18.4.7	What You May Need to Know About the Mediator Editor	18-23
18.4.7.1	Resequencing	18-23
18.4.7.2	Routing Rules	18-24
18.5	Generating a WSDL File.....	18-25
18.5.1	How to Generate a WSDL File.....	18-25
18.6	Specifying Operation or Event Subscription Properties	18-33
18.7	Modifying an Oracle Mediator Service Component.....	18-33
18.7.1	How To Modify Operations of an Oracle Mediator	18-34
18.7.2	How To Modify Event Subscriptions of an Oracle Mediator.....	18-34

19 Creating Oracle Mediator Routing Rules

19.1	Introduction to Routing Rules	19-1
19.2	Defining Routing Rules.....	19-1
19.2.1	How To Access the Routing Rules Section	19-2
19.2.2	How to Create Static Routing Rules.....	19-3
19.2.2.1	How to Specify Oracle Mediator Services or Events.....	19-4
19.2.2.2	What You May Need to Know About Echoing a Service	19-8
19.2.2.3	How to Specify Sequential or Parallel Execution.....	19-9
19.2.2.4	How to Configure Response Messages	19-10
19.2.2.5	How to Handle Multiple Callbacks	19-11
19.2.2.6	How to Handle Faults.....	19-12
19.2.2.7	How to Specify an Expression for Filtering Messages	19-14
19.2.2.8	How to Create Transformations.....	19-19
19.2.2.9	How to Assign Values	19-21
19.2.2.10	What You May Need to Know About the Assign Activity	19-25
19.2.2.11	How to Access Headers for Filters and Assignments.....	19-28
19.2.2.12	How to Use Semantic Validation	19-30
19.2.2.13	How to Use Java Callouts.....	19-32
19.2.3	How to Create Dynamic Routing Rules	19-41
19.2.4	What You May Need to Know About Using Dynamic Routing Rules.....	19-44
19.2.5	How to Define Default Routing Rules.....	19-44
19.2.5.1	Default Rule Scenarios	19-44
19.2.5.2	Default Rule Target	19-45
19.2.5.3	Default Rule: Validation, Transformation, and Assign Functionality	19-46
19.2.5.4	Default Rule: Java Callouts	19-46
19.2.5.5	Default Rule: Oracle Mediator .mplan File.....	19-46
19.3	Creating an Oracle Mediator for Routing Messages.....	19-47
19.3.1	How to Create the CustomerRouter Use Case	19-48
19.3.1.1	Task 1: How to Create an Oracle JDeveloper Application and a Project	19-48
19.3.1.2	Task 2: How to Create the CustomerRouter Oracle Mediator Service Component ...	19-49
19.3.1.3	Task 3: How to Create a File Adapter Service	19-49
19.3.1.4	Task 4: How to Create a File Adapter Reference	19-51
19.3.1.5	Task 5: How to Specify Routing Rules	19-53
19.3.1.6	Task 6: How to Create an Application Server Connection.....	19-57
19.3.1.7	Task 7: How to Deploy the CustomerRouterProject	19-57
19.3.2	Running and Monitoring the CustomerRouterProject Application.....	19-58

19.4	Creating an Asynchronous Request and Response Using Oracle Mediator	19-58
19.4.1	How to Create the AsyncMediator Use Case	19-59
19.4.1.1	Task 1: How to Create an Oracle JDeveloper Application and Project	19-59
19.4.1.2	Task 2: How to Create a Server BPEL Process	19-59
19.4.1.3	Task 3: How to Create an Oracle Mediator Service Component	19-59
19.4.1.4	Task 4: How to Create a Client BPEL Process	19-62
19.4.1.5	Task 5: How to Create the Invoke, Receive, and Assign Activities	19-63
19.4.1.6	Task 6: How to Configure an Application Server Connection	19-67
19.4.1.7	Task 7: How to Deploy the SOA Composite Application	19-67
20	Working with Multiple Part Messages in Oracle Mediator	
20.1	Introduction to Oracle Mediator Multipart Message Support	20-1
20.2	Working with Multipart Request Messages	20-2
20.2.1	How to Work with Multipart Request Messages	20-2
20.2.1.1	How to Specify Filter Expressions	20-2
20.2.1.2	How to Add Validations	20-2
20.2.1.3	How to Create Transformations	20-3
20.2.1.4	How to Assign Values	20-3
20.2.2	How to Work with Multipart Reply, Fault, and Callback Source Messages	20-3
20.2.3	How to Work with Multipart Target Messages	20-4
21	Using Oracle Mediator Error Handling	
21.1	Introduction to Oracle Mediator Error Handling	21-1
21.1.1	Fault Policies	21-1
21.1.1.1	Conditions	21-2
21.1.1.2	Actions	21-4
21.1.2	Fault Bindings	21-8
21.1.3	Error Groups in Oracle Mediator	21-9
21.2	Using Error Handling with Oracle Mediator	21-10
21.2.1	How to Use Error Handling for an Oracle Mediator Service Component	21-10
21.2.2	What Happens at Runtime	21-10
21.3	Fault Recovery Using Oracle Enterprise Manager Fusion Middleware Control	21-10
21.4	Error Handling XML Schema Definition Files	21-11
21.4.1	Schema Definition File for fault-policies.xml	21-11
21.4.2	Schema Definition File for fault-bindings.xml	21-15
22	Resequencing in Oracle Mediator	
22.1	Introduction to the Resequencer	22-1
22.1.1	Groups and Sequence IDs	22-1
22.1.2	Identification of Groups and Sequence IDs	22-2
22.2	Resequencing Order	22-2
22.2.1	Standard Resequencer	22-3
22.2.1.1	Overview of the Standard Resequencer	22-3
22.2.1.2	Information Required for Standard Resequencing	22-3
22.2.1.3	Example of the Standard Resequencer	22-3
22.2.2	FIFO Resequencer	22-4

22.2.2.1	Overview of the FIFO Resequencer	22-4
22.2.2.2	Information Required for FIFO Resequencing.....	22-4
22.2.2.3	Example of the FIFO Resequencer	22-4
22.2.3	Best Effort Resequencer	22-5
22.2.3.1	Overview of the Best Effort Resequencer.....	22-5
22.2.3.2	Information Required for Best Effort Resequencing	22-6
22.2.3.3	Example of Best Effort Resequencing Based on Maximum Rows.....	22-6
22.2.3.4	Example of Best Effort Resequencing Based on a Time Window	22-7
22.3	Configuring the Resequencer.....	22-8
22.3.1	How to Specify the Resequencing Level	22-8
22.3.2	How to Configure the Resequencing Strategy	22-9
22.4	Limitations in the Resequencer	22-12

23 Understanding Message Exchange Patterns of an Oracle Mediator

23.1	Understanding a One-way Message Exchange Pattern	23-2
23.1.1	The one.way.returns.fault Property	23-3
23.2	Understanding a Request-Reply Message Exchange Pattern.....	23-4
23.3	Understanding a Request-Reply-Fault Message Exchange Pattern	23-5
23.4	Understanding a Request-Callback Message Exchange Pattern.....	23-6
23.5	Understanding a Request-Reply-Callback Message Exchange Pattern.....	23-8
23.6	Understanding a Request-Reply-Fault-Callback Message Exchange Pattern	23-9

Part IV Using the Business Rules Service Component

24 Getting Started with Oracle Business Rules

24.1	Introduction to the Business Rule Service Component.....	24-1
24.1.1	Integrating BPEL Processes, Business Rules, and Human Tasks	24-2
24.2	Overview of Rules Designer Editor Environment	24-2
24.2.1	Application Navigator	24-3
24.2.2	Rules Designer Window	24-3
24.2.3	Structure Window	24-4
24.2.4	Business Rule Validation Log Window	24-5
24.3	Introduction to Creating and Editing Business Rules	24-5
24.3.1	How to Create Business Rules Components	24-5
24.3.2	Introduction to Working with Business Rules in Rules Designer	24-7
24.4	Adding Business Rules to a BPEL Process	24-7
24.4.1	How to Add Business Rules to a BPEL Process	24-7
24.4.2	What Happens When You Add Business Rules to a BPEL Process	24-13
24.4.3	What Happens When You Create a Business Rules Dictionary	24-14
24.4.4	What You May Need to Know About Invoking Business Rules in a BPEL Process.....	24-14
24.4.5	What You May Need to Know About Decision Component Stateful Operation .	24-14
24.5	Adding Business Rules to a SOA Composite Application	24-15
24.5.1	How to Add Business Rules to a SOA Composite Application.....	24-15
24.5.2	How to Select and Modify a Decision Function in a Business Rule Component..	24-19
24.6	Running Business Rules in a Composite Application	24-21

24.6.1	What You May Need to Know About Testing a Standalone Decision Service Component	24-21
24.7	Using Business Rules with Oracle ADF Business Components Fact Types.....	24-23

25 Using Declarative Components and Task Flows

25.1	Introduction to Declarative Components and Task Flows	25-1
25.2	Using the Oracle Business Rules Editor Declarative Component	25-2
25.2.1	Introduction to the Oracle Business Rules Editor Component	25-2
25.2.2	How to Create and Run a Sample Application by Using the Rules Editor Component...	25-6
25.2.3	How to Deploy a Rules Editor Application to a Standalone Weblogic Server.....	25-18
25.2.4	What You May Need to Know About the Custom Permissions for the Rules Editor Component	25-19
25.2.5	What You May Need to Know About the Supported Tags of the Rules Editor Component	25-20
25.3	Using the Oracle Business Rules Dictionary Editor Declarative Component	25-24
25.3.1	Introduction to the Oracle Business Rules Dictionary Component	25-24
25.3.2	How to Create and Run a Sample Application by Using the Rules Dictionary Editor Component	25-26
25.3.3	How to Deploy a Rules Dictionary Application to a Standalone Weblogic Server	25-38
25.3.4	What You May Need to Know About the Supported Attributes of the Rules Dictionary Editor Component	25-39
25.4	Using the Oracle Business Rules Dictionary Task Flow.....	25-42
25.4.1	Introduction to the Oracle Business Rules Dictionary Task Flow	25-42
25.4.2	How to Create and Run a Sample Application By Using the Rules Dictionary Editor Task Flow	25-42
25.4.3	How to Deploy a Rules Dictionary Editor Task Flow Application to a Standalone Weblogic Server	25-55
25.5	Localizing the ADF-Based Web Application	25-56

Part V Using the Human Workflow Service Component

26 Getting Started with Human Workflow

26.1	Introduction to Human Workflow	26-1
26.2	Introduction to Human Workflow Concepts.....	26-3
26.2.1	Introduction to Design and Runtime Concepts	26-3
26.2.1.1	Task Assignment and Routing	26-3
26.2.1.2	Static, Dynamic, and Rule-Based Task Assignment.....	26-6
26.2.1.3	Task Stakeholders.....	26-7
26.2.1.4	Task Deadlines	26-7
26.2.1.5	Notifications	26-8
26.2.1.6	Task Forms	26-9
26.2.1.7	Advanced Concepts	26-9
26.2.1.8	Reports and Audit Trails	26-10
26.2.2	Introduction to the Stages of Human Workflow Design.....	26-10
26.3	Introduction to Human Workflow Features	26-11

26.3.1	Human Workflow Use Cases.....	26-11
26.3.1.1	Task Assignment to a User or Role.....	26-11
26.3.1.2	Use of the Various Participant Types	26-11
26.3.1.3	Escalation, Expiration, and Delegation	26-12
26.3.1.4	Automatic Assignment and Delegation.....	26-12
26.3.1.5	Dynamic Assignment of Users Based on Task Content.....	26-13
26.3.2	Designing a Human Task from Start to Finish.....	26-13
26.3.2.1	Prerequisites	26-13
26.3.2.2	How to Create the Vacation Request Process.....	26-14
26.3.3	Additional Tutorials	26-26
26.4	Introduction to Human Workflow Architecture.....	26-27
26.4.1	Human Workflow Services	26-27
26.4.2	Use of Human Task	26-30
26.4.3	Service Engines	26-31

27 Designing Human Tasks

27.1	Introduction to Human Task Design Concepts.....	27-1
27.2	Introduction to the Modeling Process.....	27-1
27.2.1	Create a Human Task Definition.....	27-2
27.2.2	Associate the Human Task Definition with a BPEL Process.....	27-2
27.2.3	Generate the Task Form.....	27-3
27.3	Creating the Human Task Definition with the Human Task Editor.....	27-3
27.3.1	How to Create a Human Task Service Component.....	27-3
27.3.2	What Happens When You Create a Human Task Service Component	27-5
27.3.3	How to Access the Sections of the Human Task Editor.....	27-6
27.3.4	How to Specify the Title, Description, Outcome, Priority, Category, Owner, and Application Context 27-7	
27.3.4.1	Specifying a Task Title	27-8
27.3.4.2	Specifying a Task Description	27-9
27.3.4.3	Specifying a Task Outcome.....	27-9
27.3.4.4	Specifying a Task Priority	27-11
27.3.4.5	Specifying a Task Category.....	27-11
27.3.4.6	Specifying a Task Owner.....	27-11
27.3.4.7	Specifying an Application Context.....	27-17
27.3.5	How to Specify the Task Payload Data Structure.....	27-17
27.3.6	How to Assign Task Participants	27-20
27.3.6.1	Configuring the Single Participant Type	27-23
27.3.6.2	Configuring the Parallel Participant Type.....	27-31
27.3.6.3	Configuring the Serial Participant Type	27-35
27.3.6.4	Configuring the FYI Participant Type	27-38
27.3.7	How to Select a Routing Policy.....	27-39
27.3.7.1	Routing Tasks to All Participants in the Specified Order.....	27-41
27.3.7.2	Specifying Advanced Task Routing Using Business Rules.....	27-44
27.3.7.3	Using External Routing	27-49
27.3.7.4	Configuring the Error Assignee	27-51
27.3.8	How to Specify Multilingual Settings and Style Sheets.....	27-54
27.3.8.1	Specifying WordML and Other Style Sheets for Attachments	27-54

27.3.8.2	Specifying Multilingual Settings	27-54
27.3.9	How to Escalate, Renew, or End the Task.....	27-55
27.3.9.1	Introduction to Escalation and Expiration Policy.....	27-56
27.3.9.2	Specifying a Policy to Never Expire	27-57
27.3.9.3	Specifying a Policy to Expire	27-57
27.3.9.4	Extending an Expiration Policy Period	27-58
27.3.9.5	Escalating a Task Policy.....	27-58
27.3.9.6	Specifying Escalation Rules.....	27-59
27.3.9.7	Specifying a Due Date.....	27-60
27.3.10	How to Specify Participant Notification Preferences	27-60
27.3.10.1	Notifying Recipients of Changes to Task Status.....	27-62
27.3.10.2	Editing the Notification Message	27-64
27.3.10.3	Setting Up Reminders.....	27-65
27.3.10.4	Changing the Character Set Encoding.....	27-65
27.3.10.5	Securing Notifications to Exclude Details.....	27-65
27.3.10.6	Showing the Oracle BPM Worklist URL in Notifications.....	27-65
27.3.10.7	Making Email Messages Actionable	27-66
27.3.10.8	Sending Task Attachments with Email Notifications	27-66
27.3.10.9	Sending Email Notifications to Groups and Application Roles	27-66
27.3.10.10	Customizing Notification Headers	27-67
27.3.11	How to Specify Access Policies and Task Actions on Task Content.....	27-67
27.3.11.1	Specifying Access Policies on Task Content.....	27-67
27.3.12	How to Specify a Workflow Digital Signature Policy	27-71
27.3.12.1	Specifying a Certificate Authority.....	27-73
27.3.13	How to Specify Restrictions on Task Assignments	27-73
27.3.14	How to Specify Java or Business Event Callbacks	27-73
27.3.14.1	Specifying Callback Classes on Task Status	27-74
27.3.15	How to Specify Task and Routing Customizations in BPEL Callbacks	27-77
27.3.16	Disabling BPEL Callbacks	27-77
27.3.17	How to Exit the Human Task Editor and Save Your Changes	27-78
27.4	Associating the Human Task Service Component with a BPEL Process	27-78
27.4.1	How to Associate a Human Task with a BPEL Process	27-78
27.4.2	What You May Need to Know About Deleting a Wire Between a Human Task Service Component and a BPEL Process	27-79
27.4.3	How to Define the Human Task Activity Title, Initiator, Priority, and Parameter Variables	27-80
27.4.3.1	Specifying the Task Title.....	27-80
27.4.3.2	Specifying the Task Initiator and Task Priority	27-81
27.4.3.3	Specifying Task Parameters	27-81
27.4.4	How to Define the Human Task Activity Advanced Features	27-84
27.4.4.1	Specifying a Scope Name and a Global Task Variable Name.....	27-84
27.4.4.2	Specifying a Task Owner.....	27-85
27.4.4.3	Specifying an Identification Key	27-85
27.4.4.4	Specifying an Identity Context	27-85
27.4.4.5	Specifying an Application Context	27-85
27.4.4.6	Including the Task History of Other Human Tasks	27-85
27.4.5	How to View the Generated Human Task Activity	27-87

27.4.5.1	Invoking BPEL Callbacks	27-87
27.4.6	What You May Need to Know About Changing the Generated Human Task Activity... 27-89	
27.4.7	What You May Need to Know About Deleting a Partner Link Generated by a Human Task 27-90	
27.4.8	How to Define Outcome-Based Modeling.....	27-90
27.4.8.1	Specifying Payload Updates	27-90
27.4.8.2	Using Case Statements for Other Task Conclusions	27-90

28 Designing Task Forms for Human Tasks

28.1	Introduction to the Task Form	28-1
28.1.1	What You May Need to Know About Task Forms: Time Zone Conversion	28-2
28.1.2	What You May Need to Know About Task Forms: Setting Formats for Date and Time.. 28-3	
28.2	Associating the Task Flow with the Task Service	28-4
28.3	Creating an ADF Task Flow Based on a Human Task	28-4
28.3.1	How To Create an ADF Task Flow from the Human Task Editor	28-5
28.3.2	How To Create an ADF Task Flow Based on a Human Task	28-6
28.3.3	What Happens When You Create an ADF Task Flow Based on a Human Task	28-7
28.3.4	What You May Need to Know About Having Multiple ADF Task Flows That Contain the Same Element with Different Meta-attributes 28-8	
28.4	Creating a Task Form	28-9
28.4.1	How To Create an Autogenerated Task Form	28-9
28.4.2	How to Register the Library JAR File for Custom Page Templates	28-11
28.4.3	How To Create a Task Form Using the Custom Task Form Wizard	28-12
28.4.4	How To Create a Task Form Using the Complete Task with Payload Drop Handler 28-19	
28.4.5	How To Create Task Form Regions Using Individual Drop Handlers	28-26
28.4.6	How To Add the Payload to the Task Form	28-28
28.4.7	What Happens When You Create a Task Form	28-30
28.5	Refreshing Data Controls When the Task XSD Changes	28-30
28.6	Securing the Task Flow Application	28-31
28.7	Creating an Email Notification	28-31
28.7.1	How To Create an Email Notification	28-31
28.7.1.1	Creating a Task Flow with a Router	28-32
28.7.1.2	Creating an Email Notification Page	28-35
28.7.2	What Happens When You Create an Email Notification Page.....	28-38
28.7.3	What You May Need to Know About Creating an Email Notification Page.....	28-38
28.8	Deploying a Composite Application with a Task Flow	28-38
28.8.1	How To Deploy a Composite Application with a Task Flow	28-38
28.8.2	How To Redeploy the Task Form	28-39
28.8.3	How To Deploy a Task Flow as a Separate Application.....	28-39
28.8.4	How To Deploy a Task Form to a non-SOA Oracle WebLogic Server	28-39
28.8.4.1	Before Deploying the Task Form: Port Changes.....	28-39
28.8.4.2	Configuring Unique Cookie Context Paths for the Session Tracking Cookies 28-40	
28.8.4.3	Deploying oracle.soa.workflow.jar to a non-SOA Oracle WebLogic Server ..	28-40

28.8.4.4	Defining the Foreign JNDI Provider on a non-SOA Oracle WebLogic Server.....	28-42
28.8.4.5	Defining the Foreign JNDI Provider Links on a non-SOA Oracle WebLogic Server.	28-44
28.8.4.6	Including a Grant for bpm-services.jar.....	28-46
28.8.4.7	Deploying the Application.....	28-47
28.8.5	What Happens When You Deploy the Task Form	28-47
28.8.6	What You May Need to Know About Undeploying a Task Flow.....	28-47
28.9	Displaying a Task Form in the Worklist.....	28-47
28.9.1	How To Display the Task Form in the Worklist	28-48
28.10	Displaying a Task in an Email Notification	28-48
28.10.1	Changing the Text for the Worklist Application in Task Notifications.....	28-50
28.10.2	Changing the URL of the Worklist Application in Task Notifications.....	28-50
28.10.3	Showing or Hiding the URL of the Worklist Application in Task Notifications ..	28-50
28.11	Reusing the Task Flow Application with Multiple Human Tasks	28-51
28.11.1	How To Reuse the Task Flow Application with Multiple Human Tasks	28-51

29 Using Oracle BPM Worklist

29.1	Introduction to Oracle BPM Worklist	29-1
29.1.1	What You May Need To Know About Oracle BPM Worklist.....	29-3
29.2	Logging In to Oracle BPM Worklist	29-3
29.2.1	How To Log In to the Worklist.....	29-3
29.2.1.1	Enabling the weblogic User for Logging in to the Worklist.....	29-4
29.2.2	What Happens When You Log In to the Worklist.....	29-4
29.2.3	What Happens When You Change a User's Privileges While They are Logged in to Oracle BPM Worklist	29-8
29.3	Customizing the Task List Page	29-8
29.3.1	How To Filter Tasks	29-8
29.3.2	How To Create and Customize Worklist Views	29-15
29.3.3	How To Customize the Task Status Chart.....	29-19
29.3.4	How To Create a ToDo Task.....	29-20
29.3.5	How To Create a Subtask	29-21
29.4	Acting on Tasks: The Task Details Page	29-22
29.4.1	System Actions.....	29-25
29.4.2	Task History	29-26
29.4.3	How To Act on Tasks	29-28
29.4.4	How To Act on Tasks That Require a Digital Signature.....	29-36
29.5	Approving Tasks.....	29-40
29.6	Setting a Vacation Period.....	29-41
29.7	Setting Rules	29-42
29.7.1	How To Create User Rules	29-43
29.7.2	How To Create Group Rules.....	29-44
29.7.3	Assignment Rules for Tasks with Multiple Assignees.....	29-45
29.8	Using the Worklist Administration Functions	29-46
29.8.1	How To Manage Other Users' or Groups' Rules (as an Administrator).....	29-46
29.8.2	How to Specify the Login Page Realm Label.....	29-47
29.8.3	How to Specify the Resource Bundle.....	29-48

29.8.4	How to Specify the Language Locale Information	29-49
29.8.5	How to Specify a Branding Logo	29-50
29.8.6	How to Specify the Branding Title	29-51
29.8.7	How to Choose a Skin	29-51
29.8.8	How to Enable Customized Applications and Links	29-53
29.9	Specifying Notification Settings.....	29-54
29.9.1	Messaging Filter Rules	29-55
29.9.1.1	Data Types.....	29-55
29.9.1.2	Attributes	29-55
29.9.2	Rule Actions.....	29-56
29.9.3	Managing Messaging Channels.....	29-56
29.9.3.1	Viewing Your Messaging Channels.....	29-57
29.9.3.2	Creating, Editing, and Deleting a Messaging Channel.....	29-58
29.9.4	Managing Messaging Filters	29-58
29.9.4.1	Viewing Messaging Filters	29-58
29.9.4.2	Creating Messaging Filters.....	29-59
29.9.4.3	Editing a Messaging Filter.....	29-61
29.9.4.4	Deleting a Messaging Filter.....	29-61
29.10	Using Mapped Attributes (Flex Fields)	29-61
29.10.1	How To Map Attributes.....	29-62
29.10.2	Custom Mapped Attributes	29-66
29.11	Creating Worklist Reports	29-66
29.11.1	How To Create Reports	29-67
29.11.2	What Happens When You Create Reports	29-68
29.11.2.1	Unattended Tasks Report.....	29-69
29.11.2.2	Tasks Priority Report	29-70
29.11.2.3	Tasks Cycle Time Report.....	29-70
29.11.2.4	Tasks Productivity Report.....	29-71
29.12	Accessing Oracle BPM Worklist in Local Languages and Time Zones	29-72
29.12.1	Strings in Oracle BPM Worklist.....	29-72
29.12.2	How to Change the Preferred Language if the Identity Store is LDAP-Based.....	29-73
29.12.3	How to Change the Language in Which Tasks Are Displayed	29-73
29.12.4	How To Change the Language Preferences from a JAZN XML File	29-74
29.12.5	What You May Need to Know About Runtime Languages Not Displaying in the Worklist 29-75	
29.12.6	What You May Need to Know About Inconsistent Display Languages in Worklist and Embedded User's Notification Preference Interface 29-75	
29.12.7	How To Change the Time Zone Used in the Worklist.....	29-75
29.13	Creating Reusable Worklist Regions.....	29-76
29.13.1	How to Create an Application With an Embedded Reusable Worklist Region....	29-76
29.13.2	How to Set Up the Deployment Profile.....	29-79
29.13.3	How to Prepare Federated Mode Task Flows For Deployment.....	29-79
29.13.4	What You May Need to Know About Task List Task Flow	29-79
29.13.5	What You May Need to Know About Certificates Task Flow	29-83
29.13.6	What You May Need to Know About the Reports Task Flow.....	29-84
29.13.7	What You May Need to Know About Application Preferences Task Flow.....	29-86
29.13.8	What You May Need to Know About Mapped Attributes Task Flow	29-86
29.13.9	What You May Need to Know About Rules Task Flow	29-87

29.14	Java Code for Enabling Customized Applications in Worklist Application.....	29-88

30 Building a Custom Worklist Client

30.1	Introduction to Building Clients for Workflow Services	30-1
30.2	Packages and Classes for Building Clients.....	30-2
30.3	Workflow Service Clients	30-3
30.3.1	The IWorkflowServiceClient Interface	30-5
30.4	Class Paths for Clients Using SOAP.....	30-6
30.5	Class Paths for Clients Using Remote EJBs.....	30-6
30.6	Initiating a Task.....	30-7
30.6.1	Creating a Task.....	30-7
30.6.2	Creating a Payload Element in a Task.....	30-7
30.6.3	Initiating a Task Programmatically.....	30-8
30.7	Changing Workflow Standard View Definitions.....	30-9
30.8	Writing a Worklist Application Using the HelpDeskUI Sample	30-9

31 Introduction to Human Workflow Services

31.1	Introduction to Human Workflow Services.....	31-1
31.1.1	SOAP, Enterprise JavaBeans, and Java Support for the Human Workflow Services.....	31-2
31.1.1.1	Support for Foreign JNDI Names	31-3
31.1.2	Security Model for Services.....	31-4
31.1.2.1	Limitation on Propagating Identity to Workflow Services when Using SOAP Web Services	31-5
31.1.2.2	Creating Human Workflow Context on Behalf of a User.....	31-5
31.1.2.3	Obtaining the Workflow Context for a User Previously Authenticated by a JAAS Application	31-6
31.1.3	Task Service	31-6
31.1.4	Task Query Service.....	31-9
31.1.5	Identity Service.....	31-11
31.1.5.1	Identity Service Providers	31-12
31.1.6	Task Metadata Service	31-13
31.1.7	User Metadata Service	31-14
31.1.8	Task Report Service	31-16
31.1.9	Runtime Config Service	31-16
31.1.9.1	Internationalization of Attribute Labels.....	31-18
31.1.10	Evidence Store Service and Digital Signatures.....	31-19
31.1.10.1	Prerequisites	31-21
31.1.10.2	Interfaces and Methods	31-21
31.1.11	Task Instance Attributes	31-23
31.2	Notifications from Human Workflow	31-27
31.2.1	Contents of Notification.....	31-28
31.2.2	Error Message Support	31-29
31.2.3	Reliability Support.....	31-29
31.2.4	Management of Oracle Human Workflow Notification Service	31-30
31.2.5	How to Configure the Notification Channel Preferences	31-30
31.2.6	How to Configure Notification Messages in Different Languages	31-31

31.2.7	How to Send Actionable Messages	31-32
31.2.7.1	How to Send Actionable Emails for Human Tasks	31-32
31.2.8	How to Send Inbound and Outbound Attachments	31-34
31.2.9	How to Send Inbound Comments.....	31-34
31.2.10	How to Send Secure Notifications.....	31-34
31.2.11	How to Set Channels Used for Notifications.....	31-34
31.2.12	How to Send Reminders	31-34
31.2.13	How to Set Automatic Replies to Unprocessed Messages	31-35
31.2.14	How to Create Custom Notification Headers	31-36
31.3	Assignment Service Configuration	31-36
31.3.1	Dynamic Assignment and Task Escalation Functions	31-37
31.3.1.1	How to Implement a Dynamic Assignment Function	31-37
31.3.1.2	How to Configure Dynamic Assignment Functions.....	31-38
31.3.1.3	How to Configure Display Names for Dynamic Assignment Functions.....	31-39
31.3.1.4	How to Implement a Task Escalation Function	31-39
31.3.2	Dynamically Assigning Task Participants with the Assignment Service	31-39
31.3.2.1	How to Implement an Assignment Service	31-40
31.3.2.2	Example of Assignment Service Implementation.....	31-41
31.3.2.3	How to Deploy a Custom Assignment Service.....	31-43
31.3.3	Custom Escalation Function.....	31-43
31.4	Class Loading for Callbacks and Resource Bundles.....	31-44
31.5	Resource Bundles in Workflow Services	31-44
31.5.1	Task Resource Bundles	31-44
31.5.2	Global Resource Bundle – WorkflowLabels.properties	31-45
31.5.3	Worklist Client Resource Bundles.....	31-47
31.5.4	Task Detail ADF Task Flow Resource Bundles	31-47
31.5.5	Specifying Stage and Participant Names in Resource Bundles	31-47
31.5.6	Case Sensitivity in Group and Application Role Names	31-47
31.6	Introduction to Human Workflow Client Integration with Oracle WebLogic Server Services 31-48	
31.6.1	Human Workflow Services Clients.....	31-48
31.6.1.1	Task Query Service Client Code.....	31-50
31.6.1.2	Configuration Option	31-53
31.6.1.3	Client Logging.....	31-56
31.6.1.4	Configuration Migration Utility	31-56
31.6.2	Identity Propagation	31-56
31.6.2.1	Enterprise JavaBeans Identity Propagation.....	31-56
31.6.2.2	SAML Token Identity Propagation for SOAP Client	31-57
31.6.2.3	Public Key Alias.....	31-58
31.6.3	Client JAR Files	31-59
31.7	Task States in a Human Task	31-60
31.8	Database Views for Oracle Workflow.....	31-60
31.8.1	Unattended Tasks Report View	31-60
31.8.2	Task Cycle Time Report View.....	31-61
31.8.3	Task Productivity Report View	31-62
31.8.4	Task Priority Report View	31-63

32 Integrating Microsoft Excel with a Human Task

32.1	Configuring Your Environment for Invoking a BPEL Process from an Excel Workbook	32-1
32.1.1	How to Create an JDeveloper Project of the Type Web Service Data Control	32-1
32.1.2	How to Create a Dummy JSF Page	32-2
32.1.3	How to Add Desktop Integration to Your Oracle JDeveloper Project.....	32-2
32.1.4	What Happens When You Add Desktop Integration to Your JDeveloper Project .	32-2
32.1.5	How to Deploy the Web Application You Created in Step 1	32-4
32.1.6	How to Install Microsoft Excel.....	32-4
32.1.7	How to Install the Oracle ADF-Desktop Integration Plug-in	32-4
32.1.8	How to Specify the User Interface Controls and Create the Excel Workbook	32-4
32.2	Attaching Excel Workbooks to Human Task Workflow Email Notifications	32-4
32.2.1	Enabling Attachment of Excel Workbooks to Human Task Workflow Email Notifications	32-4
32.2.2	What Happens During Runtime When You Enable Attachment of Excel Workbooks to Human Task Workflow Email Notifications	32-5
32.2.3	Example: Attaching an Excel Workbook to Email Notifications	32-5
32.2.3.1	Task 1: Enable the ADF Task Flow Project with Oracle ADF-DI Capabilities .	32-5
32.2.3.2	Task 2: Set up Authentication.....	32-10
32.2.3.3	Task 3: Create a Valid Page Definition File to Be Used in the Excel Workbook	32-13
32.2.3.4	Task 4: Prepare the Excel Workbook	32-17
32.2.3.5	Task 5: Deploy the ADF Task Flow	32-23
32.2.3.6	Task 6: Test the Deployed Application	32-24

33 Configuring Task List Portlets

33.1	Introduction to Task List Portlets	33-1
33.2	Deploying the Task List Portlet Producer Application to a Portlet Server	33-2
33.2.1	Deployment Prerequisites	33-2
33.2.2	How to Deploy the Task List Portlet Producer Application	33-3
33.2.3	How to Connect the Task List Producer to the Remote SOA Server	33-3
33.2.3.1	How to Define the Foreign JNDI on the Oracle WebCenter Oracle WebLogic Server	33-3
33.2.3.2	How to Configure EJB Identity Propagation.....	33-5
33.2.3.3	How to Configure the Identity Store	33-5
33.2.4	How to Secure the Task List Portlet Producer Application Using Web Services Security	33-6
33.2.5	How to Specify the Inbound Security Policy	33-7
33.3	Creating a Portlet Consumer Application for Embedding the Task List Portlet.....	33-9
33.3.1	How To Create a Portlet Consumer Application for Embedding the Task List Portlet....	33-9
33.4	Passing Worklist Portlet Parameters.....	33-16
33.4.1	Assignment Filter Constraints	33-20
33.4.2	Example of File Containing All Column Constants	33-21

Part VI Using Binding Components

34 Getting Started with Binding Components

34.1	Introduction to Binding Components.....	34-1
34.1.1	Web Services.....	34-2
34.1.1.1	WS-AtomicTransaction Support	34-2
34.1.2	HTTP Binding Service.....	34-5
34.1.2.1	Supported Interactions	34-5
34.1.2.2	How to Configure the HTTP Binding Service.....	34-6
34.1.2.3	How to Enable Basic Authentication	34-8
34.1.3	JCA Adapters.....	34-9
34.1.3.1	AQ Adapter	34-9
34.1.3.2	Database Adapter	34-9
34.1.3.3	File Adapter	34-9
34.1.3.4	FTP Adapter	34-10
34.1.3.5	JMS Adapter	34-10
34.1.3.6	MQ Adapter	34-10
34.1.3.7	Socket Adapter.....	34-10
34.1.3.8	Third Party Adapter.....	34-10
34.1.4	Oracle Applications Adapter	34-10
34.1.5	Oracle BAM	34-11
34.1.6	Oracle B2B.....	34-11
34.1.7	ADF-BC Services.....	34-11
34.1.8	EJB Services.....	34-11
34.1.9	Direct Binding Services.....	34-12
34.2	Introduction to Integrating a Binding Component in a SOA Composite Application	34-12
34.2.1	How to Integrate a Binding Component in a SOA Composite Application.....	34-12
34.2.2	How to Use ADF Binding to Invoke a Composite Application from a JSP/Java Class....	34-13

35 Integrating Enterprise JavaBeans with SOA Composite Applications

35.1	Introduction to Enterprise JavaBeans Binding Integration with SOA Composite Applications	35-1
35.1.1	Integration Through SDO-Based EJBs	35-2
35.1.2	Integration Through Java Interfaces	35-2
35.2	Designing an SDO-Based Enterprise JavaBeans Application.....	35-3
35.2.1	How to Create SDO Objects Using the SDO Compiler.....	35-3
35.2.2	How to Create a Session Bean and Import the SDO Objects.....	35-4
35.2.3	How to Create a Profile and an EAR File	35-4
35.2.4	How to Define the SDO Types with an Enterprise JavaBeans Bean	35-4
35.2.5	How to Use Web Service Annotations	35-6
35.2.6	How to Deploy the Enterprise JavaBeans EAR File	35-8
35.3	Creating an Enterprise JavaBeans Service in Oracle JDeveloper	35-8
35.3.1	How to Integrate SDO-based Enterprise JavaBeans with SOA Composite Applications	35-8
35.3.2	How to Integrate Java Interface-based Enterprise JavaBeans with SOA Composite Applications	35-11
35.4	Designing an SDO-Based Enterprise JavaBeans Client to Invoke Oracle SOA Suite ..	35-13
35.5	Specifying Enterprise JavaBeans Roles	35-13

35.6	Configuring JNDI Access.....	35-14
35.6.1	How to Create a Foreign JNDI.....	35-14
35.6.2	How to Create a Custom CSF Map for JNDI Lookup	35-14

36 Using the Direct Binding Invocation API

36.1	Introduction to Direct Binding.....	36-1
36.2	Introduction to the Direct Binding Invocation API	36-3
36.2.1	Synchronous Direct Binding Invocation	36-4
36.2.2	Asynchronous Direct Binding Invocation.....	36-4
36.2.3	SOA Direct Address Syntax	36-5
36.2.4	SOA Transaction Propagation	36-5
36.3	Invoking a SOA Composite Application with the Invocation API	36-6
36.3.1	How to Create an Inbound Direct Binding Service	36-6
36.3.2	How to Create an Outbound Direct Binding Reference	36-8
36.3.3	How to Set an Identity for J2SE Clients Invoking Direct Binding.....	36-10
36.3.4	What You May Need to Know About Invoking SOA Composites on Hosts with the Same Server and Domain Names	36-11
36.4	Samples Using the Direct Binding Invocation API.....	36-12

Part VII Sharing Functionality Across Service Components

37 Creating Transformations with the XSLT Mapper

37.1	Introduction to the XSLT Mapper	37-1
37.1.1	Overview of XSLT Creation	37-3
37.1.2	Guidelines for Using the XSLT Mapper	37-6
37.2	Creating an XSL Map File	37-7
37.2.1	How to Create an XSL Map File in Oracle BPEL Process Manager	37-7
37.2.2	How to Create an XSL Map File from Imported Source and Target Schema Files in Oracle BPEL Process Manager	37-9
37.2.3	How to Create an XSL Map File in Oracle Mediator.....	37-11
37.2.4	What You May Need to Know About Creating an XSL Map File.....	37-14
37.2.5	What You May Need to Know About Importing a Composite with an XSL File.	37-15
37.2.6	What Happens at Runtime If You Pass a Payload Through Oracle Mediator Without Creating an XSL Map File	37-15
37.2.7	What Happens If You Receive an Empty Namespace Tag in an Output Message	37-15
37.3	Designing Transformation Maps with the XSLT Mapper	37-16
37.3.1	How to Add Additional Sources	37-16
37.3.2	How to Perform a Simple Copy by Linking Nodes.....	37-17
37.3.3	How to Set Constant Values.....	37-18
37.3.4	How to Add Functions.....	37-19
37.3.4.1	Editing Function Parameters	37-20
37.3.4.2	Chaining Functions	37-20
37.3.4.3	Using Named Templates.....	37-21
37.3.4.4	Importing User-Defined Functions.....	37-21
37.3.5	How to Edit XPath Expressions.....	37-24
37.3.6	How to Add XSLT Constructs	37-25

37.3.6.1	Using Conditional Processing with <code>xsl:if</code>	37-26
37.3.6.2	Using Conditional Processing with <code>xsl:choose</code>	37-28
37.3.6.3	Creating Loops with <code>xsl:for-each</code>	37-28
37.3.6.4	Cloning <code>xsl:for-each</code>	37-29
37.3.6.5	Applying <code>xsl:sort</code> to <code>xsl:for-each</code>	37-30
37.3.6.6	Copying Nodes with <code>xsl:copy-of</code>	37-30
37.3.6.7	Including External Templates with <code>xsl:include</code>	37-31
37.3.7	How to Automatically Map Nodes	37-32
37.3.7.1	Using Auto Mapping with Confirmation	37-33
37.3.8	What You May Need to Know About Automatic Mapping	37-34
37.3.9	How to View Unmapped Target Nodes	37-35
37.3.10	How to Generate Dictionaries	37-36
37.3.11	What You May Need to Know About Generating Dictionaries in Which Functions are Used 37-37	
37.3.12	How to Create Map Parameters and Variables	37-37
37.3.12.1	Creating a Map Parameter	37-38
37.3.12.2	Creating a Map Variable	37-38
37.3.13	How to Search Source and Target Nodes	37-39
37.3.14	How to Control the Generation of Unmapped Target Elements	37-40
37.3.15	How to Ignore Elements in the XSLT Document	37-41
37.3.16	How to Replace a Schema in the XSLT Mapper	37-41
37.3.17	How to Substitute Elements and Types in the Source and Target Trees	37-42
37.4	Testing the Map	37-45
37.4.1	How to Test the Transformation Mapping Logic	37-46
37.4.2	How to Generate Reports	37-48
37.4.2.1	Correcting Memory Errors When Generating Reports	37-49
37.4.3	How to Customize Sample XML Generation	37-50
37.5	Demonstrating Features of the XSLT Mapper	37-50
37.5.1	Opening the Application	37-51
37.5.2	Creating a New XSLT Map in the BPEL Process	37-51
37.5.3	Using Type Substitution to Map the Purchase Order Items	37-52
37.5.4	Referencing Additional Source Elements	37-53
37.5.5	Using Element Substitution to Map the Shipping Address	37-54
37.5.6	Mapping the Remaining Fields	37-55
37.5.7	Testing the Map	37-57

38 Using Business Events and the Event Delivery Network

38.1	Introduction to Business Events	38-1
38.1.1	Local and Remote Events Boundaries	38-3
38.2	Creating Business Events in Oracle JDeveloper	38-3
38.2.1	How to Create a Business Event	38-4
38.3	Subscribing to or Publishing a Business Event from an Oracle Mediator Service Component 38-6	
38.3.1	How to Subscribe to a Business Event	38-6
38.3.2	What Happens When You Create and Subscribe to a Business Event	38-8
38.3.3	What You May Need to Know About Subscribing to a Business Event	38-9
38.3.4	How to Publish a Business Event	38-9

38.3.5	How to Configure a Foreign JNDI Provider to Enable Administration Server Applications to Publish Events to the SOA Server	38-10
38.3.6	How to Configure JMS-based EDN Implementations	38-11
38.3.7	What Happens When You Publish a Business Event.....	38-12
38.4	Subscribing to or Publishing a Business Event from a BPEL Process Service Component	38-13
38.4.1	How to Subscribe to a Business Event.....	38-13
38.4.2	How to Publish a Business Event.....	38-15
38.4.3	What Happens When You Subscribe to and Publish a Business Event	38-16
38.4.4	What You May Need to Know About Subscribing to a Business Event	38-18
38.5	How to Integrate Oracle ADF Business Component Business Events with Oracle Mediator	38-18

Part VIII Completing Your Application

39 Enabling Security with Policies

39.1	Introduction to Policies	39-1
39.2	Attaching Policies to Binding Components and Service Components	39-2
39.2.1	How to Attach Policies to Binding Components and Service Components	39-2
39.2.2	How to Override Policy Configuration Property Values	39-6
39.2.2.1	Overriding Client Configuration Property Values.....	39-6
39.2.2.2	Overriding Server Configuration Property Values	39-8

40 Deploying SOA Composite Applications

40.1	Introduction to Deployment.....	40-1
40.2	Deployment Prerequisites	40-2
40.2.1	Creating the Oracle SOA Suite Schema	40-2
40.2.2	Creating a SOA Domain	40-2
40.2.3	Configuring a SOA Cluster	40-2
40.3	Understanding the Packaging Impact	40-2
40.4	Anatomy of a Composite	40-3
40.5	Preparing the Target Environment	40-3
40.5.1	Creating Data Sources and Queues.....	40-3
40.5.1.1	Script for Creation of JMS Resource and Redeployment of JMS Adapter	40-4
40.5.1.2	Script for Creation of the Database Resource and Redeployment of the Database Adapter	40-5
40.5.2	Creating Connection Factories and Connection Pooling.....	40-6
40.5.3	Enabling Security	40-6
40.5.4	Deploying Trading Partner Agreements and Task Flows.....	40-6
40.5.5	Creating an Application Server Connection.....	40-7
40.5.6	Creating a SOA-MDS Connection.....	40-7
40.6	Customizing Your Application for the Target Environment Prior to Deployment.....	40-7
40.6.1	Customizing SOA Composite Applications for the Target Environment.....	40-7
40.6.1.1	Introduction to Configuration Plans.....	40-7
40.6.1.2	Introduction to a Configuration Plan File.....	40-8
40.6.1.3	Introduction to Use Cases for a Configuration Plan	40-10
40.6.1.4	How to Create a Configuration Plan in Oracle JDeveloper	40-11

40.6.1.5	How to Create a Configuration Plan with the WLST Utility	40-14
40.6.1.6	How to Attach a Configuration Plan with ant Scripts	40-14
40.7	Deploying SOA Composite Applications	40-14
40.7.1	Deploying a Single SOA Composite in Oracle JDeveloper	40-14
40.7.1.1	How to Deploy a Single SOA Composite	40-14
40.7.1.2	What You May Need to Know About Deploying Human Task Composites with Task Flows to Partitions 40-26	
40.7.2	Deploying Multiple SOA Composite Applications in Oracle JDeveloper	40-27
40.7.2.1	How to Deploy Multiple SOA Composite Applications	40-27
40.7.3	Deploying and Using Shared Metadata Across SOA Composite Applications in Oracle JDeveloper 40-29	
40.7.3.1	How to Deploy Shared Metadata.....	40-29
40.7.3.2	How to Use Shared Metadata	40-35
40.7.4	Deploying an Existing SOA Archive in Oracle JDeveloper.....	40-39
40.7.4.1	How to Deploy an Existing SOA Archive from Oracle JDeveloper.....	40-39
40.7.5	Managing SOA Composite Applications with Scripts.....	40-40
40.7.5.1	How to Manage SOA Composite Applications with the WLST Utility	40-41
40.7.5.2	How to Manage SOA Composite Applications with ant Scripts	40-41
40.7.6	Deploying SOA Composite Applications from Oracle Enterprise Manager Fusion Middleware Control 40-60	
40.7.7	Deploying SOA Composite Applications to a Cluster	40-61
40.8	Postdeployment Configuration	40-61
40.8.1	Security	40-61
40.8.2	Updating Connections	40-61
40.8.3	Updating Data Sources and Queues	40-61
40.8.4	Attaching Policies	40-61
40.9	Testing and Troubleshooting	40-61
40.9.1	Verifying Deployment	40-61
40.9.2	Initiating an Instance of a Deployed Composite.....	40-61
40.9.3	Automating the Testing of Deployed Composites	40-62
40.9.4	Recompiling a Project After Receiving a Deployment Error.....	40-62
40.9.5	Troubleshooting Common Deployment Errors	40-62
40.9.5.1	Common Oracle JDeveloper Deployment Issues	40-62
40.9.5.2	Common Configuration Plan Issues.....	40-64
40.9.5.3	Deploying to a Managed Oracle WebLogic Server	40-64
40.9.5.4	Deploying to a Two-Way, SSL-Enabled Oracle WebLogic Server	40-65
40.9.5.5	Deploying with an Unreachable Proxy Server.....	40-65
40.9.5.6	Releasing Locks to Resolve ADF Task Form EAR File Deployment Errors ...	40-65
40.9.5.7	Increasing Memory to Recover from Compilation Errors.....	40-66

41 Automating Testing of SOA Composite Applications

41.1	Introduction to the Composite Test Framework	41-1
41.1.1	Test Cases Overview	41-1
41.1.2	Test Suites Overview	41-1
41.1.3	Emulations Overview	41-2
41.1.4	Assertions Overview	41-2
41.2	Introduction to the Components of a Test Suite.....	41-3

41.2.1	Process Initiation.....	41-3
41.2.2	Emulations.....	41-3
41.2.3	Assertions.....	41-4
41.2.4	Message Files.....	41-5
41.3	Creating Test Suites and Test Cases.....	41-5
41.3.1	How to Create Test Suites and Test Cases.....	41-5
41.4	Creating the Contents of Test Cases.....	41-8
41.4.1	How to Initiate Inbound Messages.....	41-9
41.4.2	How to Emulate Outbound Messages.....	41-11
41.4.3	How to Emulate Callback Messages.....	41-14
41.4.4	How to Emulate Fault Messages.....	41-16
41.4.5	How to Create Assertions.....	41-17
41.4.5.1	Creating Assertions on a Part Section, Nonleaf Element, or Entire XML Document 41-18	
41.4.5.2	Creating Assertions on a Leaf Element.....	41-21
41.4.6	What You May Need to Know About Assertions.....	41-23
41.5	Deploying and Running a Test Suite.....	41-23

Part IX Advanced Topics

42 Managing Large Documents and Large Numbers of Instances

42.1	Best Practices for Handling Large Documents.....	42-1
42.1.1	Use Cases for Handling Large Documents.....	42-1
42.1.1.1	Passing Binary Objects as Base64-Encoded Text in XML Payloads.....	42-1
42.1.1.2	End-to-End Streaming with Attachments.....	42-2
42.1.1.3	Adding MTOM Attachments to Web Services.....	42-9
42.1.1.4	Processing Large XML with Repeating Constructs.....	42-10
42.1.1.5	Processing Large XML Documents with Complex Structures.....	42-11
42.1.2	Limitations on Concurrent Processing of Large Documents.....	42-12
42.1.2.1	Opaque Schema for Processing Large Payloads.....	42-12
42.1.3	General Tuning Recommendations.....	42-12
42.1.3.1	General Recommendations.....	42-12
42.1.3.2	Setting Audit Levels from Oracle Enterprise Manager for Large Payload Processing 42-13	
42.1.3.3	Using the Assign Activity in Oracle BPEL Process Manager/Oracle Mediator..... 42-13	
42.1.3.4	Using XSLT Transformations on Large Payloads (For Oracle BPEL Process Manager) 42-14	
42.1.3.5	Using XSLT Transformations for Repeating Structures.....	42-15
42.1.3.6	Processing Large Documents in Oracle B2B.....	42-16
42.1.3.7	Using XPath Functions to Write Large XSLT/XQuery Output to a File System..... 42-18	
42.2	Best Practices for Handling Large Metadata.....	42-18
42.2.1	Boundary on the Processing of Large Numbers of Activities in a BPEL Process .	42-18
42.2.2	Using Large Numbers of Activities in BPEL Processes (Without FlowN).....	42-19
42.2.3	Using Large Numbers of Activities in BPEL Processes (With FlowN).....	42-19
42.2.4	Using a Flow With Multiple Sequences.....	42-19

42.2.5	Using a Flow with One Sequence.....	42-19
42.2.6	Using a Flow with No Sequence.....	42-20
42.2.7	Large Numbers of Oracle Mediators in a Composite.....	42-20
42.2.8	Importing Large Data Sets in Oracle B2B.....	42-20
42.3	Best Practices for Handling Large Numbers of Instances.....	42-20
42.3.1	Instance and Rejected Message Deletion with the Purge Script.....	42-20
42.3.2	Improving the Loading of Pages in Oracle Enterprise Manager Fusion Middleware Control	42-20

43 Customizing SOA Composite Applications

43.1	Introduction to Customizing SOA Composite Applications.....	43-1
43.2	Creating the Customizable Composite	43-2
43.2.1	How to Create the Customizable Composite	43-2
43.2.2	How to Create Customization Classes	43-3
43.2.3	How to Add XSD or WSDL Files.....	43-4
43.2.4	How to Search for Customized Activities in a BPEL Process	43-4
43.2.5	What You May Need to Know About Editing Artifacts in a Customized Composite.....	43-5
43.2.6	What You May Need to Know About Resolving Validation Errors in Oracle JDeveloper	43-5
43.2.7	What You May Need to Know About Resolving a Sequence Conflict	43-6
43.2.8	What You May Need to Know About Compiling and Deploying a Customized Application	43-7
43.3	Customizing the Vertical Application	43-7
43.3.1	How to Customize the Vertical Application.....	43-7
43.4	Customizing the Customer Version.....	43-9
43.4.1	How to Customize the Customer Version	43-9
43.5	Upgrading the Composite	43-11
43.5.1	How to Upgrade the Core Application Team Composite	43-11
43.5.2	How to Upgrade the Vertical Application Team Composite.....	43-11
43.5.3	How to Upgrade the Customer Composite	43-12

44 Working with Domain Value Maps

44.1	Introduction to Domain Value Maps	44-1
44.1.1	Domain Value Map Features	44-2
44.1.1.1	Qualifier Support.....	44-2
44.1.1.2	Qualifier Order Support	44-3
44.1.1.3	One-to-Many Mapping Support	44-4
44.2	Creating Domain Value Maps.....	44-4
44.2.1	How to Create Domain Value Maps.....	44-4
44.2.2	What Happens When You Create a Domain Value Map	44-5
44.3	Editing a Domain Value Map.....	44-7
44.3.1	How to Add Columns to a Domain Value Map	44-7
44.3.2	How to Add Rows to a Domain Value Map.....	44-7
44.4	Using Domain Value Map Functions.....	44-8
44.4.1	Understanding Domain Value Map Functions	44-8
44.4.1.1	dvm:lookupValue.....	44-8

44.4.1.2	dvm:lookupValue1M	44-9
44.4.2	How to Use Domain Value Map Functions in Transformations	44-9
44.4.3	How to Use Domain Value Map Functions in XPath Expressions.....	44-11
44.4.4	What Happens at Runtime	44-12
44.5	Creating a Domain Value Map Use Case for a Hierarchical Lookup	44-12
44.5.1	How to Create the HierarchicalValue Use Case	44-13
44.5.1.1	Task 1: How to Create an Oracle JDeveloper Application and a Project	44-13
44.5.1.2	Task 2: How to Create a Domain Value Map	44-13
44.5.1.3	Task 3: How to Create a File Adapter Service	44-15
44.5.1.4	Task 4: How to Create ProcessOrders Oracle Mediator Component	44-16
44.5.1.5	Task 5: How to Create a File Adapter Reference	44-17
44.5.1.6	Task 6: How to Specify Routing Rules	44-18
44.5.1.7	Task 7: How to Configure an Application Server Connection	44-21
44.5.1.8	Task 8: How to Deploy the Composite Application.....	44-22
44.5.2	How to Run and Monitor the HierarchicalValue Application.....	44-22
44.6	Creating a Domain Value Map Use Case For Multiple Values.....	44-22
44.6.1	How to Create the Multivalue Use Case	44-22
44.6.1.1	Task 1: How to Create an Oracle JDeveloper Application and Project	44-23
44.6.1.2	Task 2: How to Create a Domain Value Map	44-23
44.6.1.3	Task 3: How to Create a File Adapter Service	44-24
44.6.1.4	Task 4: How to Create the LookupMultiplevaluesMediator Oracle Mediator	44-26
44.6.1.5	Task 5: How to Create a File Adapter Reference	44-26
44.6.1.6	Task 6: How to Specify Routing Rules	44-27
44.6.1.7	Task 7: How to Configure an Application Server Connection	44-30
44.6.1.8	Task 8: How to Deploy the Composite Application.....	44-30
44.6.2	How to Run and Monitor the Multivalue Application	44-31

45 Using Oracle SOA Composer with Domain Value Maps

45.1	Introduction to Oracle SOA Composer	45-1
45.1.1	How to Log in to Oracle SOA Composer	45-2
45.2	Viewing Domain Value Maps at Runtime	45-3
45.2.1	How To View Domain Value Maps at Runtime	45-3
45.3	Editing Domain Value Maps at Runtime	45-4
45.3.1	How to Edit Domain Value Maps at Runtime	45-4
45.3.1.1	Adding Rows	45-5
45.3.1.2	Editing Rows	45-5
45.3.1.3	Deleting Rows	45-5
45.4	Saving Domain Value Maps at Runtime	45-5
45.4.1	How to Save Domain Value Maps at Runtime	45-5
45.5	Committing Changes at Runtime.....	45-5
45.5.1	How to Commit Changes at Runtime	45-6
45.6	Detecting Conflicts.....	45-6

46 Working with Cross References

46.1	Introduction to Cross References.....	46-1

46.2	Introduction to Cross Reference Tables.....	46-2
46.3	Creating and Modifying Cross Reference Tables.....	46-4
46.3.1	How to Create Cross Reference Metadata	46-4
46.3.2	What Happens When You Create a Cross Reference	46-6
46.3.3	How to Create Custom Database Tables.....	46-7
46.3.4	How to Add an End System to a Cross Reference Table	46-8
46.4	Populating Cross Reference Tables	46-9
46.4.1	About the xref:populateXRefRow Function	46-10
46.4.2	About the xref:populateXRefRow1M Function.....	46-12
46.4.3	How to Populate a Column of a Cross Reference Table	46-13
46.5	Looking Up Cross Reference Tables	46-15
46.5.1	About the xref:lookupXRef Function.....	46-15
46.5.2	About the xref:lookupXRef1M Function.....	46-16
46.5.3	About the xref:lookupPopulatedColumns Function.....	46-17
46.5.4	How to Look Up a Cross Reference Table for a Value.....	46-18
46.6	Deleting a Cross Reference Table Value	46-19
46.6.1	How to Delete a Cross Reference Table Value	46-20
46.7	Creating and Running the Cross Reference Use Case.....	46-22
46.7.1	How to Create the Use Case.....	46-22
46.7.1.1	Task 1: How to Configure the Oracle Database and Database Adapter	46-22
46.7.1.2	Task 2: How to Create an Oracle JDeveloper Application and a Project	46-23
46.7.1.3	Task 3: How to Create a Cross Reference	46-24
46.7.1.4	Task 4: How to Create a Database Adapter Service	46-25
46.7.1.5	Task 5: How to Create EBS and SBL External References	46-27
46.7.1.6	Task 6: How to Create the Logger File Adapter External Reference	46-29
46.7.1.7	Task 7: How to Create an Oracle Mediator Service Component.....	46-31
46.7.1.8	Task 8: How to Specify Routing Rules for an Oracle Mediator Service Component .	46-32
46.7.1.9	Task 9: How to Specify Routing Rules for the Common Oracle Mediator	46-42
46.7.1.10	Task 10: How to Configure an Application Server Connection	46-53
46.7.1.11	Task 11: How to Deploy the Composite Application.....	46-53
46.7.2	How to Run and Monitor the XrefCustApp Application	46-53
46.8	Creating and Running Cross Reference for 1M Functions	46-54
46.8.1	How to Create the Use Case.....	46-54
46.8.1.1	Task 1: How to Configure the Oracle Database and Database Adapter	46-54
46.8.1.2	Task 2: How to Create an Oracle JDeveloper Application and a Project	46-55
46.8.1.3	Task 3: How to Create a Cross Reference	46-56
46.8.1.4	Task 4: How to Create a Database Adapter Service	46-57
46.8.1.5	Task 5: How to Create an EBS External Reference	46-59
46.8.1.6	Task 6: How to Create a Logger File Adapter External Reference.....	46-61
46.8.1.7	Task 7: How to Create an Oracle Mediator Service Component.....	46-62
46.8.1.8	Task 8: How to Specify Routing Rules for an Oracle Mediator Component..	46-63
46.8.1.9	Task 9: How to Specify Routing Rules for the Common Oracle Mediator	46-67
46.8.1.10	Task 10: How to Configure an Application Server Connection	46-72
46.8.1.11	Task 11: How to Deploy the Composite Application.....	46-72

47 Defining Composite Sensors

47.1	Introduction to Composite Sensors	47-1
47.1.1	Restrictions on Use of Composite Sensors	47-1
47.2	Adding Composite Sensors	47-2
47.2.1	How to Add Composite Sensors	47-2
47.2.2	How to Add a Variable	47-6
47.2.3	How to Add an Expression	47-6
47.2.4	How to Add a Property	47-7
47.3	Monitoring Composite Sensor Data During Runtime.....	47-8

48 Using Two-Layer Business Process Management (BPM)

48.1	Introduction to Two-Layer Business Process Management	48-1
48.2	Creating a Phase Activity	48-3
48.2.1	How to Create a Phase Activity.....	48-3
48.2.2	What Happens When You Create a Phase Activity.....	48-4
48.2.3	What Happens at Runtime When You Create a Phase Activity.....	48-5
48.2.4	What You May Need to Know About Creating a Phase Activity	48-5
48.3	Creating the Dynamic Routing Decision Table	48-6
48.3.1	How to Create the Dynamic Routing Decision Table	48-6
48.3.2	What Happens When You Create the Dynamic Routing Decision Table	48-7
48.4	Use Case: Two-Layer BPM	48-7
48.4.1	Designing the SOA Composite.....	48-8
48.4.2	Creating a Phase Activity	48-10
48.4.3	Creating and Editing the Dynamic Routing Decision Table	48-11
48.4.4	Adding Assign Activities to the BPEL Process Model	48-12
48.4.5	Deploying and Testing the Sample.....	48-14

49 Integrating the Spring Framework in SOA Composite Applications

49.1	Introduction to the Spring Service Component.....	49-1
49.2	Integration of Java and WSDL-Based Components in the Same SOA Composite Application 49-2	
49.2.1	Java and WSDL-Based Integration Example	49-2
49.2.2	Using Callbacks with the Spring Framework.....	49-4
49.3	Creating a Spring Service Component in Oracle JDeveloper.....	49-5
49.3.1	How to Create a Spring Service Component in Oracle JDeveloper	49-6
49.3.2	What You May Need to Know About Java Class Errors During Java-to-WSDL Conversions 49-17	
49.4	Defining Custom Spring Beans Through a Global Spring Context.....	49-17
49.4.1	How to Define Custom Spring Beans Through a Global Spring Context	49-17
49.5	Using the Predefined Spring Beans.....	49-17
49.5.1	IHeaderHelperBean.java Interface for headerHelperBean.....	49-18
49.5.2	IInstanceHelperBean.java Interface for instancerHelperBean	49-18
49.5.3	ILoggerBean.java Interface for loggerBean.....	49-19
49.5.4	How to Reference Predefined Spring Beans in the Spring Context File.....	49-20
49.6	Spring Service Component Integration in the Fusion Order Demo.....	49-21
49.6.1	How to Use EJBs with Java Vector Type Parameters	49-26

49.7	JAXB and OXM Support	49-28
49.7.1	Extended Mapping Files	49-29

Part X Using Oracle Business Activity Monitoring

50 Integrating Oracle BAM with SOA Composite Applications

50.1	Introduction to Integrating Oracle BAM with SOA Composite Applications.....	50-1
50.2	Configuring Oracle BAM Adapter	50-2
50.3	Using Oracle BAM Monitor Express With BPEL Processes	50-2
50.3.1	How to Access BPEL Designer Monitor View	50-4
50.3.2	How to Configure Activity Monitors	50-5
50.3.3	How To Create BPEL Process Monitoring Objects	50-6
50.3.4	How to Configure Counters.....	50-7
50.3.5	How to Configure Intervals	50-9
50.3.6	How to Configure Business Indicators.....	50-11
50.3.7	How to Add Existing Monitoring Objects to Activities	50-14
50.3.8	How To Configure BPEL Process Monitors for Deployment	50-15
50.3.9	What You Need to Know About Using the Monitor Express Dashboard	50-18
50.3.10	What You Need To Know About Monitor Express Data Objects	50-18
50.3.10.1	Understanding the COMPONENT Data Object	50-19
50.3.10.2	Understanding the COUNTER Data Object	50-20
50.3.10.3	Understanding the INTERVAL Data Object	50-21
50.3.10.4	Understanding Business Indicator Data Objects	50-23
50.3.10.5	Troubleshooting.....	50-24
50.4	Creating a Design Time Connection to an Oracle BAM Server	50-25
50.4.1	How to Create a Connection to an Oracle BAM Server	50-25
50.5	Using Oracle BAM Adapter in a SOA Composite Application	50-26
50.5.1	How to Use Oracle BAM Adapter in a SOA Composite Application	50-26
50.6	Using Oracle BAM Adapter in a BPEL Process.....	50-27
50.6.1	How to Use Oracle BAM Adapter in a BPEL Process.....	50-27
50.7	Integrating BPEL Sensors Using Oracle BAM Sensor Action	50-29
50.7.1	How to Create a Sensor.....	50-29
50.7.2	How to Create an Oracle BAM Sensor Action	50-30
50.8	Integrating SOA Applications and Oracle BAM Using Enterprise Message Resources	50-33

51 Using Oracle BAM Data Control

51.1	Introduction to Oracle BAM Data Control.....	51-1
51.2	Creating Projects That Can Use Oracle BAM Data Controls.....	51-2
51.3	Creating Oracle BAM Server Connections.....	51-2
51.3.1	How to Modify Oracle BAM Data Control Connections to Oracle BAM Servers ..	51-2
51.3.1.1	How to Associate a BAM Data Control with a New Oracle BAM Connection	51-3
51.4	Exposing Oracle BAM with Oracle ADF Data Controls	51-4
51.4.1	How to Create Oracle BAM Data Controls.....	51-4
51.4.2	What Happens in Your Project When You Create an Oracle BAM Data Control ..	51-4
51.4.2.1	How an Oracle BAM Data Control Appears in the Data Controls Panel	51-5
51.5	Creating Oracle BAM Data Control Queries	51-5

51.5.1	How to Choose a Query Type.....	51-6
51.5.2	How to Create Parameters	51-7
51.5.3	How to Pass Values to Parameters.....	51-8
51.5.4	How to Create Calculated Fields.....	51-9
51.5.4.1	Creating Groups in Calculated Fields	51-10
51.5.5	How to Select, Organize, and Sort Fields	51-11
51.5.6	How to Create Filters	51-11
51.5.6.1	How to Create Filter Headers	51-11
51.5.6.2	How to Create Filter Entries	51-12
51.5.6.3	Entering Comparison Values.....	51-13
51.5.6.4	Using Active Now	51-14
51.5.7	How to Select and Organize Groups	51-15
51.5.7.1	How to Configure Time Groups and Time Series	51-15
51.5.8	How to Create Aggregates	51-16
51.5.9	How to Modify the Query	51-16
51.6	Using Oracle BAM Data Controls in ADF Pages	51-16
51.6.1	How to Use an Oracle BAM Data Control in a JSF Page	51-17
51.7	Deploying Applications With Oracle BAM Data Controls.....	51-17
51.7.1	How to Deploy to Oracle WebLogic Server in Development Mode.....	51-18
51.7.2	How to Deploy to a Production Mode Oracle WebLogic Server	51-18

52 Defining and Managing Oracle BAM Data Objects

52.1	Introduction to Oracle BAM Data Objects	52-1
52.2	Defining Data Objects.....	52-2
52.2.1	How to Define a Data Object	52-2
52.2.2	How to Add Columns to a Data Object.....	52-3
52.2.3	How to Add Lookup Columns to a Data Object.....	52-4
52.2.4	How to Add Calculated Columns to a Data Object.....	52-5
52.2.5	How to Add Time Stamp Columns to a Data Object	52-6
52.2.6	What You May Need to Know About System Data Objects	52-6
52.2.7	What You May Need to Know About Oracle Data Integrator Data Objects	52-6
52.3	Creating Permissions on Data Objects	52-6
52.3.1	How to Create Permissions on a Data Object.....	52-6
52.3.2	How to Add a Group of Users.....	52-7
52.3.3	How to Copy Permissions from Other Data Objects.....	52-7
52.4	Viewing Existing Data Objects.....	52-8
52.4.1	How to View Data Object General Information.....	52-8
52.4.2	How to View Data Object Layouts.....	52-9
52.4.3	How to View Data Object Contents	52-9
52.5	Using Data Object Folders	52-10
52.5.1	How to Create Folders	52-10
52.5.2	How to Open Folders	52-11
52.5.3	How to Set Folder Permissions.....	52-11
52.5.4	How to Move Folders.....	52-12
52.5.5	How to Rename Folders	52-12
52.5.6	How to Delete Folders	52-12
52.6	Creating Security Filters.....	52-13

52.6.1	How to Create a Security Filter.....	52-13
52.6.2	How to Copy Security Filters from Other Data Objects	52-15
52.7	Creating Dimensions	52-15
52.7.1	How to Create a Dimension.....	52-15
52.7.2	How to Create a Time Dimension	52-16
52.8	Renaming and Moving Data Objects	52-17
52.8.1	How to Rename a Data Object.....	52-17
52.8.2	How to Move a Data Object	52-17
52.9	Creating Indexes	52-17
52.9.1	How to Create an Index.....	52-18
52.10	Clearing Data Objects.....	52-18
52.10.1	How to Clear a Data Object.....	52-18
52.11	Deleting Data Objects	52-18
52.11.1	How to Delete a Data Object.....	52-18

53 Creating Oracle BAM Enterprise Message Sources

53.1	Introduction to Enterprise Message Sources	53-1
53.2	Creating Enterprise Message Sources.....	53-2
53.2.1	How to Create an Enterprise Message Source.....	53-2
53.2.2	How to Configure DateTime Specification	53-7
53.2.3	How to Use Advanced XML Formatting	53-10
53.3	Using Enterprise Message Sources.....	53-11
53.3.1	How to Edit, Copy, and Delete Enterprise Message Sources.....	53-11
53.3.2	How to Start and Stop Enterprise Message Sources.....	53-11
53.3.3	How to Subscribe and Unsubscribe Enterprise Message Sources.....	53-12
53.3.4	How to Test Enterprise Message Sources	53-12
53.3.5	How to Refresh Enterprise Message Sources	53-12
53.3.6	How to Monitor Enterprise Message Source Metrics.....	53-12
53.4	Using Foreign JMS Providers.....	53-13
53.5	Use Case: Creating an EMS Against Oracle Streams AQ JMS Provider.....	53-14
53.5.1	Creating a JMS Topic in AQ-JMS	53-14
53.5.2	Creating a Data Source in Oracle WebLogic Server	53-15
53.5.3	Creating a Foreign JMS Server.....	53-16
53.5.4	Defining an EMS in Oracle BAM Architect	53-17
53.5.5	Inserting and Updating Records in the SQL Table.....	53-18

54 Using Oracle Data Integrator With Oracle BAM

54.1	Introduction to Using the Oracle Data Integrator With Oracle Business Activity Monitoring 54-1	
54.2	Installing the Oracle Data Integrator Integration Files.....	54-2
54.2.1	How to Install Integration Files Using the Script.....	54-2
54.2.2	How to Manually Install Integration Files	54-4
54.2.3	Using the Logs.....	54-7
54.3	Using Oracle BAM Knowledge Modules	54-7
54.4	Creating the Oracle BAM Target	54-13
54.4.1	How to Create the Oracle BAM Target	54-13
54.5	Reverse Engineering the Oracle BAM Schema.....	54-14

54.6	Updating the Oracle Data Integrator External Data Source Definition	54-14
54.6.1	How to Update the Oracle Data Integrator External Data Source Definitions.....	54-15
54.7	Launching Oracle Data Integrator Scenarios From Oracle BAM Alerts.....	54-15
54.8	Running Oracle Data Integrator Agent as a Daemon or a Microsoft Windows Service With Oracle BAM Embedded	54-15

55 Creating External Data Sources

55.1	Introduction to External Data Sources	55-1
55.2	Creating External Data Sources	55-2
55.2.1	How to Create an External Data Source	55-2
55.2.2	What You May Need to Know About Oracle Data Integrator External Data Sources.....	55-2
55.2.3	How to Edit an External Data Source	55-2
55.2.4	How to Delete an External Data Source	55-3
55.3	External Data Source Example.....	55-3
55.4	Use Case: Creating an EDS Against Oracle Business Intelligence Enterprise Edition ..	55-4

56 Using Oracle BAM Web Services

56.1	Introduction to Oracle BAM Web Services	56-1
56.2	Using the DataObjectOperations Web Services	56-2
56.2.1	How to Use the DataObjectOperations Web Services.....	56-2
56.3	Using the DataObjectDefinition Web Service	56-3
56.3.1	How to Use the DataObjectDefinition Web Service	56-3
56.4	Using the ManualRuleFire Web Service	56-4
56.4.1	How to Use the ManualRuleFire Web Service	56-4
56.5	Using the ICommand Web Service	56-4
56.5.1	How to Use the ICommand Web Service.....	56-5

57 Creating Oracle BAM Alerts

57.1	Introduction to Creating Alerts	57-1
57.2	Creating Alert Rules	57-2
57.2.1	How to Create an Alert Rule.....	57-2
57.2.2	How to Activate Alerts	57-3
57.2.3	How to Modify Alert Rules	57-4
57.2.4	How to Delete an Alert	57-4
57.3	Creating Alert Rules From Templates	57-4
57.3.1	How to Create Alert Rules From Templates.....	57-4
57.4	Creating Alert Rules With Messages	57-5
57.4.1	How to Create an Alert Rule With a Message.....	57-5
57.5	Creating Complex Alerts	57-6
57.5.1	How to Create a Dependent Rule.....	57-6
57.6	Using Alert History	57-6
57.6.1	How to View Alert History	57-7
57.6.2	How to Clear Alert History	57-8
57.7	Launching Alerts by Invoking Web Services.....	57-8
57.8	Calling an External Action.....	57-8

57.9	Sending Alerts to External E-mail Accounts.....	57-9

58 Using ICommand

58.1	Introduction to ICommand	58-1
58.2	Executing ICommand.....	58-1
58.3	Specifying the Command and Option Syntax	58-2
58.3.1	How to Specify the Security Credentials.....	58-2
58.3.2	How to Specify the Command.....	58-3
58.3.3	How to Specify Object Names	58-3
58.3.4	How to Specify Multiple Parameter Targets	58-4
58.4	Using Command-line-only Parameters.....	58-5
58.5	Running ICommand Remotely	58-6

Part XI Using Oracle User Messaging Service

59 Oracle User Messaging Service

59.1	Introduction to User Messaging Service.....	59-1
59.1.1	Components.....	59-2
59.1.2	Architecture	59-2

60 Sending and Receiving Messages using the User Messaging Service EJB API

60.1	Introduction to the UMS Java API.....	60-1
60.1.1	Creating a Java EE Application Module.....	60-2
60.2	Creating a UMS Client Instance.....	60-2
60.2.1	Creating a MessagingEJBClient Instance Using a Programmatic or Declarative Approach 60-2	
60.2.2	API Reference for Class MessagingClientFactory.....	60-3
60.3	Sending a Message.....	60-3
60.3.1	Creating a Message.....	60-3
60.3.1.1	Creating a Plaintext Message.....	60-3
60.3.1.2	Creating a Multipart/ Alternative Message (with Text/Plain and Text/HTML Parts) 60-3	
60.3.1.3	Creating Delivery Channel-Specific Payloads in a Single Message for Recipients with Different Delivery Types 60-4	
60.3.2	API Reference for Class MessageFactory	60-4
60.3.3	API Reference for Interface Message	60-5
60.3.4	API Reference for Enum DeliveryType.....	60-5
60.3.5	Addressing a Message	60-5
60.3.5.1	Types of Addresses	60-5
60.3.5.2	Creating Address Objects	60-5
60.3.5.3	Creating a Recipient with a Failover Address.....	60-5
60.3.5.4	API Reference for Class AddressFactory	60-6
60.3.5.5	API Reference for Interface Address	60-6
60.3.6	Retrieving Message Status.....	60-6
60.3.6.1	Synchronous Retrieval of Message Status	60-6

60.3.6.2	Asynchronous Notification of Message Status	60-6
60.4	Receiving a Message.....	60-6
60.4.1	Registering an Access Point	60-6
60.4.2	Synchronous Receiving.....	60-7
60.4.3	Asynchronous Receiving.....	60-7
60.4.4	Message Filtering	60-7
60.5	Using the UMS Enterprise JavaBeans Client API to Build a Client Application.....	60-8
60.5.1	Overview of Development	60-9
60.5.2	Configuring the Email Driver	60-9
60.5.3	Using JDeveloper 11g to Build the Application	60-9
60.5.3.1	Opening the Project.....	60-9
60.5.4	Deploying the Application	60-11
60.5.5	Testing the Application.....	60-11
60.6	Using the UMS Enterprise JavaBeans Client API to Build a Client Echo Application	60-13
60.6.1	Overview of Development	60-14
60.6.2	Configuring the Email Driver	60-14
60.6.3	Using JDeveloper 11g to Build the Application	60-15
60.6.3.1	Opening the Project.....	60-15
60.6.4	Deploying the Application	60-18
60.6.5	Testing the Application.....	60-18
60.7	Creating a New Application Server Connection.....	60-20

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

61.1	Introduction to the UMS Java API.....	61-2
61.2	Creating a UMS Client Instance and Specifying Runtime Parameters	61-2
61.2.1	API Reference for Class MessagingClientFactory.....	61-3
61.3	Sending a Message.....	61-4
61.3.1	Creating a Message.....	61-4
61.3.1.1	Creating a Plaintext Message.....	61-4
61.3.1.2	Creating a Multipart/Alternative Message (with Text/Plain and Text/HTML Parts) 61-4	
61.3.1.3	Creating Delivery Channel-Specific Payloads in a Single Message for Recipients with Different Delivery Types 61-5	
61.3.2	API Reference for Class MessagingFactory	61-5
61.3.3	API Reference for Interface Message	61-6
61.3.4	API Reference for Enum DeliveryType.....	61-6
61.3.5	Addressing a Message	61-6
61.3.5.1	Types of Addresses	61-6
61.3.5.2	Creating Address Objects.....	61-6
61.3.5.3	Creating a Recipient with a Failover Address.....	61-6
61.3.5.4	API Reference for Class MessagingFactory	61-7
61.3.5.5	API Reference for Interface Address	61-7
61.3.6	User Preference Based Messaging	61-7
61.4	Retrieving Message Status.....	61-7
61.4.1	Synchronous Retrieval of Message Status.....	61-7
61.4.2	Asynchronous Receiving of Message Status	61-8

61.4.2.1	Creating a Listener Programmatically.....	61-8
61.4.2.2	Default Status Listener.....	61-8
61.4.2.3	Per Message Status Listener.....	61-8
61.5	Receiving a Message.....	61-9
61.5.1	Registering an Access Point	61-9
61.5.2	Synchronous Receiving.....	61-9
61.5.3	Asynchronous Receiving.....	61-10
61.5.3.1	Creating a Listener Programmatically.....	61-10
61.5.3.2	Default Message Listener	61-10
61.5.3.3	Per Access Point Message Listener	61-11
61.5.4	Message Filtering	61-11
61.6	Configuring for a Cluster Environment	61-11
61.7	Configuring Security	61-12
61.8	Threading Model.....	61-12
61.8.1	Listener Threading	61-13
61.9	Using the UMS Client API to Build a Client Application	61-13
61.9.1	Overview of Development	61-14
61.9.2	Configuring the Email Driver	61-14
61.9.3	Using JDeveloper 11g to Build the Application.....	61-14
61.9.3.1	Opening the Project.....	61-14
61.9.4	Deploying the Application	61-16
61.9.5	Testing the Application.....	61-17
61.10	Using the UMS Client API to Build a Client Echo Application	61-19
61.10.1	Overview of Development	61-20
61.10.2	Configuring the Email Driver	61-20
61.10.3	Using JDeveloper 11g to Build the Application.....	61-20
61.10.3.1	Opening the Project.....	61-20
61.10.4	Deploying the Application.....	61-22
61.10.5	Testing the Application.....	61-23
61.11	Creating a New Application Server Connection.....	61-24

62 Sending and Receiving Messages using the User Messaging Service Web Service API

62.1	Introduction to the UMS Web Service API	62-1
62.2	Creating a UMS Client Instance and Specifying Runtime Parameters	62-2
62.3	Sending a Message.....	62-3
62.3.1	Creating a Message.....	62-4
62.3.1.1	Creating a Plaintext Message.....	62-4
62.3.1.2	Creating a Multipart/Mixed Message (with Text and Binary Parts).....	62-4
62.3.1.3	Creating a Multipart/Alternative Message (with Text/Plain and Text/HTML Parts) 62-4	
62.3.1.4	Creating Delivery Channel-Specific Payloads in a Single Message for Recipients with Different Delivery Types 62-5	
62.3.2	API Reference for Interface Message	62-6
62.3.3	API Reference for Enum DeliveryType.....	62-6
62.3.4	Addressing a Message	62-6
62.3.4.1	Types of Addresses	62-6

62.3.4.2	Creating Address Objects.....	62-6
62.3.4.3	Creating a Recipient with a Failover Address.....	62-7
62.3.4.4	Recipient Types.....	62-7
62.3.4.5	API Reference for Class MessagingFactory	62-7
62.3.4.6	API Reference for Interface Address	62-7
62.3.5	User Preference Based Messaging	62-7
62.4	Retrieving Message Status.....	62-8
62.4.1	Synchronous Retrieval of Message Status.....	62-8
62.4.2	Asynchronous Receiving of Message Status	62-8
62.4.2.1	Creating a Listener Programmatically.....	62-8
62.4.2.2	Publish the Callback Service	62-9
62.4.2.3	Stop a Dynamically Published Endpoint	62-9
62.4.2.4	Registration.....	62-9
62.5	Receiving a Message.....	62-9
62.5.1	Registering an Access Point	62-10
62.5.2	Synchronous Receiving.....	62-10
62.5.3	Asynchronous Receiving.....	62-11
62.5.3.1	Creating a Listener Programmatically.....	62-11
62.5.3.2	Default Message Listener	62-11
62.5.3.3	Per Access Point Message Listener	62-12
62.5.4	Message Filtering	62-12
62.6	Configuring for a Cluster Environment	62-12
62.7	Configuring Security	62-13
62.7.1	Client and Server Security	62-13
62.7.2	Listener/Callback Security.....	62-13
62.8	Threading Model.....	62-14
62.9	Sample Chat Application with Web Services APIs.....	62-14
62.9.1	Overview.....	62-14
62.9.1.1	Provided Files	62-14
62.9.2	Running the Pre-Built Sample	62-15
62.9.3	Testing the Sample.....	62-17
62.10	Creating a New Application Server Connection.....	62-20

63 Parlay X Web Services Multimedia Messaging API

63.1	Introduction to Parlay X Messaging Operations.....	63-1
63.2	Send Message Interface.....	63-2
63.2.1	sendMessage Operation.....	63-2
63.2.2	getMessageDeliveryStatus Operation	63-3
63.3	Receive Message Interface	63-4
63.3.1	getReceivedMessages Operation.....	63-4
63.3.2	getMessage Operation.....	63-5
63.3.3	getMessageURIs Operation.....	63-5
63.4	Oracle Extension to Parlay X Messaging.....	63-6
63.4.1	ReceiveMessageManager Interface	63-6
63.4.1.1	startReceiveMessage Operation	63-7
63.4.1.2	stopReceiveMessage Operation.....	63-7
63.5	Parlay X Messaging Client API and Client Proxy Packages.....	63-8

63.6	Sample Chat Application with Parlay X APIs	63-8
63.6.1	Overview	63-9
63.6.1.1	Provided Files	63-9
63.6.2	Running the Pre-Built Sample	63-9
63.6.3	Testing the Sample.....	63-12
63.6.4	Creating a New Application Server Connection.....	63-14

64 User Messaging Preferences

64.1	Introduction to User Messaging Preferences.....	64-1
64.1.1	Terminology	64-1
64.1.2	Configuration of Notification Delivery Preferences.....	64-2
64.1.3	Delivery Preference Rules	64-2
64.1.3.1	Data Types.....	64-2
64.1.3.2	System Terms	64-3
64.1.3.3	Business Terms.....	64-3
64.1.4	Rule Actions.....	64-4
64.2	How to Manage Messaging Channels	64-5
64.2.1	Creating a Channel	64-5
64.2.2	Editing a Channel	64-6
64.2.3	Deleting a Channel	64-7
64.2.4	Setting a Default Channel.....	64-7
64.3	Creating Contact Rules using Filters.....	64-7
64.3.1	Creating Filters.....	64-9
64.3.2	Editing a Filter	64-10
64.3.3	Deleting a Filter	64-10
64.4	Configuring Settings.....	64-10

Part XII Appendices

A BPEL Process Activities and Services

A.1	Introduction to Activities and Components	A-1
A.2	Introduction to BPEL 1.1 and 2.0 Activities	A-2
A.2.1	Tabs Common to Many Activities.....	A-4
A.2.1.1	Annotations Tab	A-4
A.2.1.2	Assertions Tab.....	A-4
A.2.1.3	Correlations Tab	A-5
A.2.1.4	Documentation Tab.....	A-5
A.2.1.5	Headers Tab.....	A-5
A.2.1.6	Properties Tab	A-5
A.2.1.7	Skip Condition Tab.....	A-6
A.2.1.8	Source and Targets Tabs.....	A-6
A.2.1.9	Timeout Tab	A-6
A.2.2	Assign Activity.....	A-6
A.2.3	Assert Activity.....	A-9
A.2.4	Bind Entity Activity	A-10
A.2.5	Compensate Activity.....	A-11

A.2.6	CompensateScope Activity	A-12
A.2.7	Create Entity Activity.....	A-13
A.2.8	Dehydrate Activity	A-13
A.2.9	Email Activity.....	A-14
A.2.10	Empty Activity	A-15
A.2.11	Exit Activity	A-16
A.2.12	Flow Activity	A-16
A.2.13	FlowN Activity.....	A-17
A.2.14	forEach Activity	A-18
A.2.15	If Activity	A-19
A.2.16	IM Activity.....	A-20
A.2.17	Invoke Activity.....	A-20
A.2.18	Java Embedding Activity.....	A-21
A.2.19	Partner Link Activity.....	A-22
A.2.20	Phase Activity.....	A-23
A.2.21	Pick Activity	A-24
A.2.22	Receive Activity	A-27
A.2.23	Receive Signal Activity	A-28
A.2.24	Remove Entity Activity.....	A-28
A.2.25	RepeatUntil Activity.....	A-29
A.2.26	Replay Activity.....	A-30
A.2.27	Reply Activity.....	A-30
A.2.28	Rethrow Activity.....	A-31
A.2.29	Scope Activity.....	A-32
A.2.30	Sequence Activity	A-33
A.2.31	Signal Activity	A-34
A.2.32	SMS Activity	A-35
A.2.33	Switch Activity.....	A-35
A.2.34	Terminate Activity.....	A-36
A.2.35	Throw Activity	A-37
A.2.36	Transform Activity	A-37
A.2.37	User Notification Activity	A-38
A.2.38	Validate Activity	A-39
A.2.39	Voice Activity.....	A-40
A.2.40	Wait Activity	A-40
A.2.41	While Activity	A-41
A.3	Introduction to BPEL Services	A-42
A.3.1	ADF-BC Service.....	A-42
A.3.2	AQ Adapter	A-42
A.3.3	Oracle B2B.....	A-42
A.3.4	Oracle BAM Adapter.....	A-42
A.3.5	Database Adapter	A-43
A.3.6	Direct Binding Service.....	A-43
A.3.7	EJB Service	A-43
A.3.8	File Adapter	A-43
A.3.9	FTP Adapter	A-43
A.3.10	HTTP Binding.....	A-43

A.3.11	JMS Adapter	A-44
A.3.12	MQ Adapter.....	A-44
A.3.13	Oracle Applications	A-44
A.3.14	Socket Adapter.....	A-44
A.3.15	Third Party Adapter	A-44
A.3.16	Web Service	A-44
A.4	Publishing and Browsing the Oracle Service Registry	A-44
A.4.1	How to Publish a Business Service	A-45
A.4.2	How to Create a Connection to the Registry	A-45
A.4.3	How to Configure a SOA Project to Invoke a Service from the Registry	A-46
A.4.3.1	Dynamically Resolving the SOAP Endpoint Location	A-47
A.4.3.2	Dynamically Resolving the WSDL Endpoint Location.....	A-48
A.4.3.3	Resolving Endpoints	A-49
A.4.4	How To Configure the Inquiry URL, UDDI Service Key, and Endpoint Address for Runtime A-51	
A.4.4.1	Changing Endpoint Locations in the Registry Control.....	A-52
A.4.4.2	Publishing WSDLs from Multiple SOA Partitions	A-54
A.4.5	How to Publish WSDLs to UDDI for Multiple Partitions.....	A-54
A.5	Providing Design-time Governance with the Oracle Enterprise Repository.....	A-55
A.6	Validating When Loading a Process Diagram.....	A-55

B XPath Extension Functions

B.1	SOA XPath Extension Functions.....	B-1
B.1.1	Database Functions.....	B-1
B.1.1.1	lookup-table.....	B-1
B.1.1.2	query-database.....	B-2
B.1.1.3	sequence-next-val	B-2
B.1.2	Date Functions.....	B-3
B.1.2.1	add-dayTimeDuration-to-dateTime	B-3
B.1.2.2	current-date	B-3
B.1.2.3	current-dateTime	B-4
B.1.2.4	current-time	B-4
B.1.2.5	day-from-dateTime	B-4
B.1.2.6	format-dateTime	B-5
B.1.2.7	hours-from-dateTime	B-5
B.1.2.8	implicit-timezone.....	B-5
B.1.2.9	minutes-from-dateTime.....	B-6
B.1.2.10	month-from-dateTime	B-6
B.1.2.11	seconds-from-dateTime	B-6
B.1.2.12	subtract-dayTimeDuration-from-dateTime.....	B-6
B.1.2.13	timezone-from-dateTime.....	B-7
B.1.2.14	year-from-dateTime	B-7
B.1.3	Mathematical Functions.....	B-7
B.1.3.1	abs	B-8
B.1.4	String Functions	B-8
B.1.4.1	compare.....	B-8
B.1.4.2	compare-ignore-case	B-9

B.1.4.3	create-delimited-string.....	B-9
B.1.4.4	ends-with	B-9
B.1.4.5	format-string	B-10
B.1.4.6	get-content-as-string	B-10
B.1.4.7	get-content-from-file-function	B-10
B.1.4.8	get-localized-string.....	B-11
B.1.4.9	index-within-string.....	B-11
B.1.4.10	last-index-within-string	B-12
B.1.4.11	left-trim	B-12
B.1.4.12	lower-case	B-13
B.1.4.13	matches.....	B-13
B.1.4.14	right-trim.....	B-13
B.1.4.15	upper-case.....	B-14
B.2	BPEL XPath Extension Functions	B-14
B.2.1	addQuotes.....	B-14
B.2.2	authenticate.....	B-14
B.2.3	appendToList.....	B-16
B.2.4	copyList	B-16
B.2.5	countNodes.....	B-17
B.2.6	doc	B-17
B.2.7	doStreamingTranslate	B-18
B.2.8	doTranslateFromNative.....	B-18
B.2.9	doTranslateToNative.....	B-18
B.2.10	doXSLTransform.....	B-19
B.2.11	doXSLTransformForDoc.....	B-20
B.2.12	formatDate	B-21
B.2.13	generateGUID	B-21
B.2.14	getApplicationName	B-21
B.2.15	getAttachmentContent.....	B-22
B.2.16	getComponentName	B-22
B.2.17	getComponentInstanceID.....	B-22
B.2.18	getCompositeName	B-22
B.2.19	getCompositeInstanceID	B-23
B.2.20	getCompositeURL	B-23
B.2.21	getContentAsString	B-23
B.2.22	getConversationId	B-23
B.2.23	getCreator	B-24
B.2.24	getCurrentDate.....	B-24
B.2.25	getCurrentDateTime	B-24
B.2.26	getCurrentTime	B-24
B.2.27	getDomainId	B-25
B.2.28	getECID	B-25
B.2.29	getElement	B-25
B.2.30	getFaultAsString	B-25
B.2.31	getFaultName.....	B-26
B.2.32	getGroupIdsFromGroupAlias	B-26
B.2.33	getInstanceId	B-26

B.2.34	getNodeValue.....	B-26
B.2.35	getNodes	B-27
B.2.36	getOwnerDocument	B-27
B.2.37	getParentComponentInstanceID	B-27
B.2.38	getPreference	B-28
B.2.39	getProcessId.....	B-28
B.2.40	getProcessOwnerId	B-28
B.2.41	getProcessURL	B-28
B.2.42	getProcessVersion.....	B-29
B.2.43	getUserAliasId.....	B-29
B.2.44	getUserIdsFromGroupAlias.....	B-29
B.2.45	setCompositeInstanceTitle	B-29
B.2.46	instanceOf	B-30
B.2.47	integer.....	B-30
B.2.48	listUsers.....	B-30
B.2.49	lookupUser	B-31
B.2.50	parseEscapedXML	B-32
B.2.51	parseXML.....	B-32
B.2.52	processXQuery	B-32
B.2.53	processXSLT	B-33
B.2.54	processXSLTAttachment	B-36
B.2.55	processXSQL.....	B-37
B.2.56	readBinaryFromFile.....	B-37
B.2.57	readFile.....	B-38
B.2.58	search.....	B-38
B.2.59	writeBinaryToFile	B-40
B.2.60	BPEL Extension Functions in BPEL 1.1 and BPEL 2.0.....	B-40
B.2.60.1	getLinkStatus.....	B-40
B.2.60.2	getVariableData	B-41
B.2.60.3	getVariableProperty (For BPEL 1.1).....	B-41
B.2.60.4	getVariableProperty (For BPEL 2.0).....	B-42
B.2.61	Utility Functions	B-42
B.2.61.1	batchProcessActive.....	B-42
B.2.61.2	batchProcessCompleted	B-43
B.2.61.3	format	B-43
B.2.61.4	genEmptyElem.....	B-43
B.2.61.5	getChildElement	B-44
B.2.61.6	getMessage	B-44
B.2.61.7	max-value-among-nodeset.....	B-44
B.2.61.8	min-value-among-nodeset	B-45
B.2.61.9	square-root.....	B-45
B.2.61.10	translateFromNative	B-45
B.2.61.11	translateToNative	B-46
B.2.61.12	translateFromNativeAttachment	B-46
B.2.61.13	translateToNativeAttachment	B-46
B.3	Oracle Mediator XPath Extension Functions.....	B-47
B.3.1	doStreamingTranslate	B-47

B.3.2	doTranslateFromNative.....	B-47
B.3.3	doTranslateToNative.....	B-48
B.3.4	getAttachmentContent.....	B-49
B.3.5	getComponentInstanceID.....	B-49
B.3.6	getComponentName.....	B-49
B.3.7	getCompositeInstanceID.....	B-50
B.3.8	getCompositeName.....	B-50
B.3.9	getHeader.....	B-50
B.3.10	getECID.....	B-51
B.3.11	getParentComponentInstanceID.....	B-51
B.3.12	setCompositeInstanceTitle.....	B-51
B.3.13	translateFromNativeAttachment.....	B-51
B.3.14	translateToNativeAttachment.....	B-52
B.4	Advanced Functions.....	B-52
B.4.1	create-nodeset-from-delimited-string.....	B-52
B.4.2	generate-guid.....	B-53
B.4.3	lookupPopulatedColumns.....	B-53
B.4.4	lookupValue.....	B-54
B.4.5	lookupValue1M.....	B-54
B.4.6	lookupXRef.....	B-55
B.4.7	lookupXRef1M.....	B-55
B.4.8	lookup-xml.....	B-56
B.4.9	markForDelete.....	B-56
B.4.10	populateXRefRow.....	B-57
B.4.11	populateXRefRow1M.....	B-57
B.5	Workflow Service Functions.....	B-58
B.5.1	clearTaskAssignees.....	B-58
B.5.2	createWordMLDocument.....	B-58
B.5.3	getNotificationProperty.....	B-58
B.5.4	getNumberOfTaskApprovals.....	B-59
B.5.5	getPreviousTaskApprover.....	B-59
B.5.6	getTaskAttachmentByIndex.....	B-59
B.5.7	getTaskAttachmentByName.....	B-60
B.5.8	getTaskAttachmentContents.....	B-60
B.5.9	getTaskAttachmentsCount.....	B-60
B.5.10	getTaskResourceBundleString.....	B-61
B.5.11	wfDynamicGroupAssign.....	B-61
B.5.12	wfDynamicUserAssign.....	B-62
B.5.13	Identity Service Functions.....	B-63
B.5.13.1	getDefaultRealmName.....	B-63
B.5.13.2	getGroupProperty.....	B-63
B.5.13.3	getManager.....	B-63
B.5.13.4	getReportees.....	B-64
B.5.13.5	getSupportedRealmNames.....	B-64
B.5.13.6	getUserProperty.....	B-64
B.5.13.7	getUserRoles.....	B-65
B.5.13.8	getusersinapprole.....	B-66

B.5.13.9	getUsersInGroup	B-66
B.5.13.10	isUserInRole	B-66
B.5.13.11	lookupGroup	B-67
B.5.13.12	lookupUser	B-67
B.6	Building XPath Expressions in Oracle JDeveloper	B-67
B.6.1	How to Use the Expression Builder	B-67
B.6.2	Introduction to the XPath Building Assistant.....	B-69
B.6.3	How to Use the XPath Building Assistant	B-69
B.6.4	Using the XPath Building Assistant in the XSLT Mapper	B-70
B.6.5	Function Parameter Tool Tips.....	B-72
B.6.6	Syntactic and Semantic Validation.....	B-72
B.6.7	Creating Expressions with Free Form Text and XPath Expressions	B-73
B.7	Creating User-Defined XPath Extension Functions.....	B-74
B.7.1	How to Implement User-Defined XPath Extension Functions	B-77
B.7.1.1	How to Implement Functions for the XSLT Mapper	B-77
B.7.1.2	How to Implement Functions for All Other Components	B-77
B.7.2	How to Configure User-Defined XPath Extension Functions.....	B-78
B.7.3	How to Deploy User-Defined Functions to Runtime	B-80

C Deployment Descriptor Properties

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

D Understanding Sensor Public Views and the Sensor Actions XSD

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

E Oracle BAM Web Services Operations

E.1	DataObjectOperations10131	E-1
E.1.1	Batch	E-1
E.1.1.1	Request Message.....	E-1
E.1.2	Delete	E-2
E.1.2.1	Request Message.....	E-2
E.1.3	Insert	E-2
E.1.3.1	Request Message.....	E-2
E.1.4	Update	E-3
E.1.4.1	Request Message.....	E-3
E.1.5	Upsert	E-3

E.1.5.1	Request Message.....	E-3
E.2	DataObjectOperationsByName.....	E-4
E.2.1	Delete.....	E-4
E.2.1.1	Request Message.....	E-4
E.2.2	Get.....	E-4
E.2.2.1	Request Message.....	E-4
E.2.3	Insert.....	E-5
E.2.3.1	Request Message.....	E-5
E.2.4	Update.....	E-5
E.2.4.1	Request Message.....	E-5
E.2.5	Upsert.....	E-5
E.2.5.1	Request Message.....	E-6
E.3	DataObjectOperationsByID.....	E-6
E.3.1	Batch.....	E-6
E.3.1.1	Request Message.....	E-6
E.3.2	Delete.....	E-7
E.3.2.1	Request Message.....	E-7
E.3.3	Insert.....	E-7
E.3.3.1	Request Message.....	E-7
E.3.4	Update.....	E-8
E.3.4.1	Request Message.....	E-8
E.3.5	Upsert.....	E-8
E.3.5.1	Request Message.....	E-8
E.4	DataObjectDefinition Operations.....	E-9
E.4.1	Create.....	E-9
E.4.1.1	Request Message.....	E-9
E.4.1.2	Response Message.....	E-11
E.4.2	Delete.....	E-11
E.4.2.1	Request Message.....	E-11
E.4.2.2	Response Message.....	E-11
E.4.3	Get.....	E-11
E.4.3.1	Request Message.....	E-11
E.4.3.2	Response Message.....	E-11
E.4.4	Update.....	E-12
E.4.4.1	Request Message.....	E-12
E.4.4.2	Response Message.....	E-12
E.5	ManualRuleFire Operations.....	E-12
E.5.1	FireRuleByName.....	E-13
E.5.1.1	Request Message.....	E-13
E.5.1.2	Response Message.....	E-13

F Oracle BAM Alert Rule Options

F.1	Events.....	F-1
F.1.1	In a specific amount of time.....	F-1
F.1.2	At a specific time today.....	F-1
F.1.3	On a certain day at a specific time.....	F-2
F.1.4	Every interval between two times.....	F-2

F.1.5	Every date interval starting on certain date at a specific time	F-2
F.1.6	When a report changes	F-2
F.1.7	When a data field changes in data object	F-3
F.1.8	When a data field in a report meets specified conditions.....	F-3
F.1.9	When a data field in a data object meets specified conditions.....	F-4
F.1.10	When this rule is launched	F-5
F.2	Conditions	F-5
F.2.1	If it is between two times	F-5
F.2.2	If It is between two days	F-5
F.2.3	If it is a particular day of the week.....	F-5
F.3	Actions	F-5
F.3.1	Send a report via email	F-6
F.3.2	Send a message via email	F-6
F.3.3	Send a report via email and escalate to another user after a specific amount of time	F-6
F.3.4	Send a parameterized message.....	F-6
F.3.5	Send a parameterized message for every matching row in a data object	F-10
F.3.6	Launch a rule	F-11
F.3.7	Launch rule if an action fails	F-11
F.3.8	Delete rows from a Data Object.....	F-11
F.3.9	Call a Web Service	F-11
F.3.9.1	How to Use Call a Web Service: An Example	F-13
F.3.10	Run an Oracle Data Integrator Scenario.....	F-14
F.3.11	Call an External Action	F-14
F.4	Frequency Constraint	F-14

G Oracle BAM ICommand Operations and File Formats

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

H Normalized Message Properties

H.1	Introduction to Normalized Messages	H-1
H.2	Oracle BPEL Process Manager Properties.....	H-2
H.3	Oracle Web Services Addressing Properties.....	H-3
H.4	Manipulating Normalized Message Properties with bpelx Extensions	H-4

H.4.1	BPEL 1.1 bpelx Extensions Syntax.....	H-4
H.4.2	BPEL 2.0 bpelx Extensions Syntax.....	H-5

I Interfaces Implemented By Rules Dictionary Editor Task Flow

I.1	The MetadataDetails Interface	I-1
I.1.1	The getDocument Method	I-2
I.1.2	The getRelatedDocument Method	I-3
I.1.3	The setDocument Method	I-3
I.2	The NLSPreferences Interface	I-4

J Oracle User Messaging Service Applications

J.1	Send Message to User Specified Channel	J-1
J.1.1	Overview	J-2
J.1.1.1	Provided Files	J-2
J.1.2	Installing and Configuring SOA and User Messaging Service.....	J-2
J.1.2.1	Updating Addresses in Your LDAP User Profile	J-2
J.1.3	Building the Sample	J-3
J.1.4	Creating a New Application Server Connection.....	J-10
J.1.5	Deploying the Project	J-11
J.1.6	Configuring User Messaging Preferences.....	J-12
J.1.7	Testing the Sample.....	J-13
J.1.7.1	Verifying the Execution of Sending the Email	J-13
J.2	Send Email with Attachments.....	J-14
J.2.1	Overview	J-14
J.2.1.1	Provided Files	J-15
J.2.2	Installing and Configuring SOA and User Messaging Service.....	J-15
J.2.2.1	Updating Addresses in Your LDAP User Profile	J-15
J.2.3	Running the Pre-Built Sample	J-16
J.2.4	Testing the Sample.....	J-17
J.2.4.1	Verifying the Execution	J-18
J.2.5	Building the Sample	J-18
J.2.5.1	Sending Text Content with base64 Encoding.....	J-27
J.2.6	Creating a New Application Server Connection.....	J-28

K Oracle SOA Suite Properties Road Map

K.1	Oracle BPEL Process Manager Deployment Descriptor Properties	K-1
K.2	Normalized Message Header Properties.....	K-2
K.2.1	Oracle JCA Adapter Message Header Properties	K-2
K.2.2	Oracle BPEL Process Manager and Oracle Web Services Addressing Message Header Properties K-2	
K.2.3	Oracle B2B Message Header Properties	K-3
K.3	SOA Composite Application Properties.....	K-3
K.4	Fault Policy and Adapter Rejected Message Properties.....	K-4
K.5	Oracle B2B System Properties	K-4
K.6	Oracle Enterprise Manager Fusion Middleware Control Property Pages.....	K-4
K.6.1	SOA Infrastructure Properties	K-5

K.6.2	Oracle BPEL Process Manager	K-5
K.6.3	Human Workflow Notification and Task Service.....	K-5
K.6.4	Oracle Mediator	K-6
K.6.5	Cross References	K-6
K.6.6	Oracle B2B.....	K-6
K.6.7	Service and Reference Binding Component Properties	K-6
K.7	System MBean Browser Properties	K-7
K.7.1	SOA Infrastructure Properties	K-7
K.7.2	Oracle BPEL Process Manager Properties.....	K-8
K.7.3	Oracle Mediator Properties	K-8
K.7.4	Human Workflow Notification and Task Service Properties	K-8
K.7.5	Oracle Service Registry WSDL URL Caching Configuration.....	K-9

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 developing applications with Oracle SOA Suite.

Documentation Accessibility

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/support/contact.html> or visit <http://www.oracle.com/accessibility/support.html> if you are hearing impaired.

Related Documents

For more information, see the following Oracle resources:

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

Printed documentation is available for sale in the Oracle Store at

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

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

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

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

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

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

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

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

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

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

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

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

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

Conventions

The following text conventions are used in this document:

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

Part I

Introduction to Oracle SOA Suite

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

This part contains the following chapters:

- [Chapter 1, "Introduction to Building Applications with Oracle SOA Suite"](#)
- [Chapter 2, "Developing SOA Composite Applications with Oracle SOA Suite"](#)
- [Chapter 3, "Introduction to the SOA Sample Application"](#)

Introduction to Building Applications with Oracle SOA Suite

This chapter describes the architecture and key functionality of Oracle SOA Suite.

This chapter includes the following sections:

- [Section 1.1, "Introduction to Service-Oriented Architecture"](#)
- [Section 1.2, "Introduction to Services"](#)
- [Section 1.3, "Introduction to Oracle SOA Suite"](#)
- [Section 1.4, "Standards Used by Oracle SOA Suite to Enable SOA"](#)
- [Section 1.5, "Service Component Architecture within SOA Composite Applications"](#)
- [Section 1.6, "Runtime Behavior of a SOA Composite Application"](#)
- [Section 1.7, "Approaches for Designing SOA Composite Applications"](#)
- [Section 1.8, "Learning Oracle SOA Suite"](#)

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 predicting change in a global business environment, to rapidly respond to competitors, and to best exploit organizational assets for growth. In response to these challenges, leading companies are adopting service-oriented architecture (SOA) to deliver on these requirements by overcoming the complexity of their application and IT environments.

SOA provides an enterprise architecture that supports building connected enterprise applications to provide solutions to business problems. 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 Services

SOA separates business functions into distinct units, or services. A SOA application reuses services to automate a business process.

A standard interface and message structure define services. The most widely used mechanism are web services standards. These standards include the Web Service Description Language (WSDL) file for service interface definition and XML Schema Documents (XSD) for message structure definition. These XML standards are easily

exchanged using standard protocols. Because standards for web services use a standard document structure, they enable existing systems to interoperate regardless of the choice of operating system and computer language used for service implementation.

When designing a SOA approach, you create a service portfolio plan to identify common functionality to use as a service within the business process. By creating and maintaining a plan, you ensure that existing services and applications are reused or repurposed whenever possible. This plan also reduces the time spent in creating needed functionality for the application.

1.3 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 Oracle SOA Suite benefit from common capabilities, including a single deployment, management, and tooling model, 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
- Web services management and security
- Business rules
- Events framework
- Business activity monitoring

1.4 Standards Used by Oracle SOA Suite to Enable SOA

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 a SOA composite application. The specification for the SCA standard is maintained by the Organization for the Advancement of Structured Information Standards (OASIS) through the Open Composite Services Architecture (CSA) Member Section:
<http://www.oasis-open.org>
- Service Data Objects (SDO)
Specifies a standard data method and can modify business data regardless of how it is physically accessed. Knowledge is not required about how to access a particular back-end data source to use SDO in a 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. BPEL versions 1.1 and 2.0 are supported.

- **XSL Transformations (XSLT)**

Processes XML documents and transforms document data from one XML schema to another.

- **Java Connector Architecture (JCA)**

Provides a Java technology solution to the problem of connectivity between the many application servers in Enterprise Information Systems (EIS).

- **Java Messaging Service (JMS)**

Provides a messaging standard that allows application components based on the Java 2 Platform, Enterprise Edition (Java EE) to access business logic distributed among heterogeneous systems.

- **Web Services Description Language (WSDL) file**

Provides the entry points into a 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.

1.5 Service Component Architecture within SOA Composite Applications

Oracle SOA Suite uses the SCA standard as a way to assemble service components into a SOA composite application. 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 a SOA composite application. In the SCA environment, service components are the building blocks of applications.

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

The key benefits of SCA include the following:

- **Loose coupling**

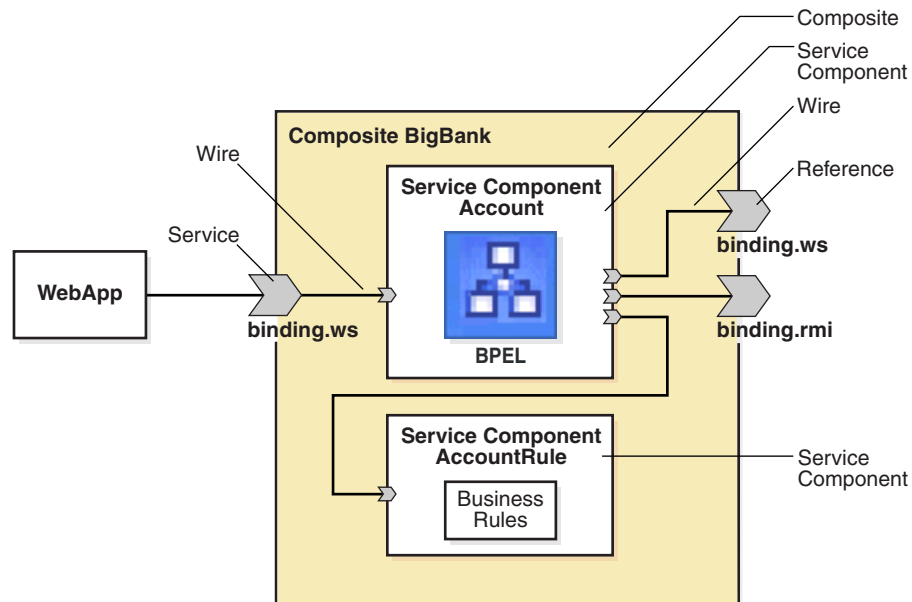
Service components integrate with other service components without needing to know how other service components are implemented.

- Flexibility
Service components can easily be replaced by other service components.
- Services invocation
Services can be invoked either synchronously or asynchronously.
- Productivity
Service components are easily integrated to create a SOA composite application.
- Easy Maintenance and Debugging
Service components can be easily maintained and debugged when an issue is encountered.

A SOA composite is an assembly of services, service components, and references designed and deployed in a single application. Wiring between the services, service components, and references enables message communication. The details for a composite are stored in the `composite.xml` file.

Figure 1–1 provides an example of a composite that includes an inbound service binding component, a BPEL process service component (named `Account`), a business rules service component (named `AccountRule`), and two outbound reference binding components.

Figure 1–1 Simple SOA Composite Architecture



1.5.1 Service Components

Service components are the building blocks that you use to construct a SOA composite application.

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 in a single composite.

- BPEL processes provide process orchestration and storage of a synchronous or an 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 enable you to design a business decision based on rules.
- Human tasks provide workflow modeling that describes the tasks for users or groups to perform as part of an end-to-end business process flow.
- Mediators route events (messages) between different components
- Spring enables you to integrate Java interfaces into SOA composite applications

1.5.2 Binding Components

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

- Services provide the outside world with an entry point to the SOA composite application. The WSDL file of the service advertises its capabilities to external applications. These capabilities are used for contacting the SOA composite application components. The binding connectivity of the service describes the protocols that can communicate with the service, for example, SOAP/HTTP or a JCA adapter.
- References enable messages to be sent from the SOA composite application to external services in the outside world.

[Table 1–1](#) lists and describes the binding components provided by Oracle SOA Suite.

Table 1–1 Binding Components Provided by Oracle SOA Suite

Binding Components	Description
Web service (SOAP over HTTP)	Use for connecting to standards-based services using SOAP over HTTP.
JCA adapters	Use for integrating services and references with technologies (for example, databases, file systems, FTP servers, messaging: JMS, IBM WebSphere MQ, and so on) and applications (Oracle E-Business Suite, PeopleSoft, and so on). This includes the AQ adapter, database adapter, file adapter, FTP adapter, JMS adapter, MQ adapter, and Socket adapter.
B2B binding component	Use for browsing B2B metadata in the MDS repository and selecting document definitions.
ADF-BC service	Use for connecting Oracle Application Development Framework (ADF) applications using SDO with the SOA platform.
Oracle Applications	Use for integrating the Oracle Applications adapter with Oracle applications.
BAM adapter	Use for integrating Java EE applications with Oracle BAM Server to send data, and also use as a reference binding component in a SOA composite application.
EJB service	Use for integrating SDO parameters or Java interfaces with Enterprise JavaBeans.
Direct binding service	Use to invoke a SOA composite application and exchange messages over a remote method invocation (RMI) in the inbound direction and to invoke an Oracle Service Bus (OSB) flow or another SOA composite application in the outbound direction.
HTTP binding	Use to integrate SOA composite applications with HTTP binding.

1.5.3 Wires

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

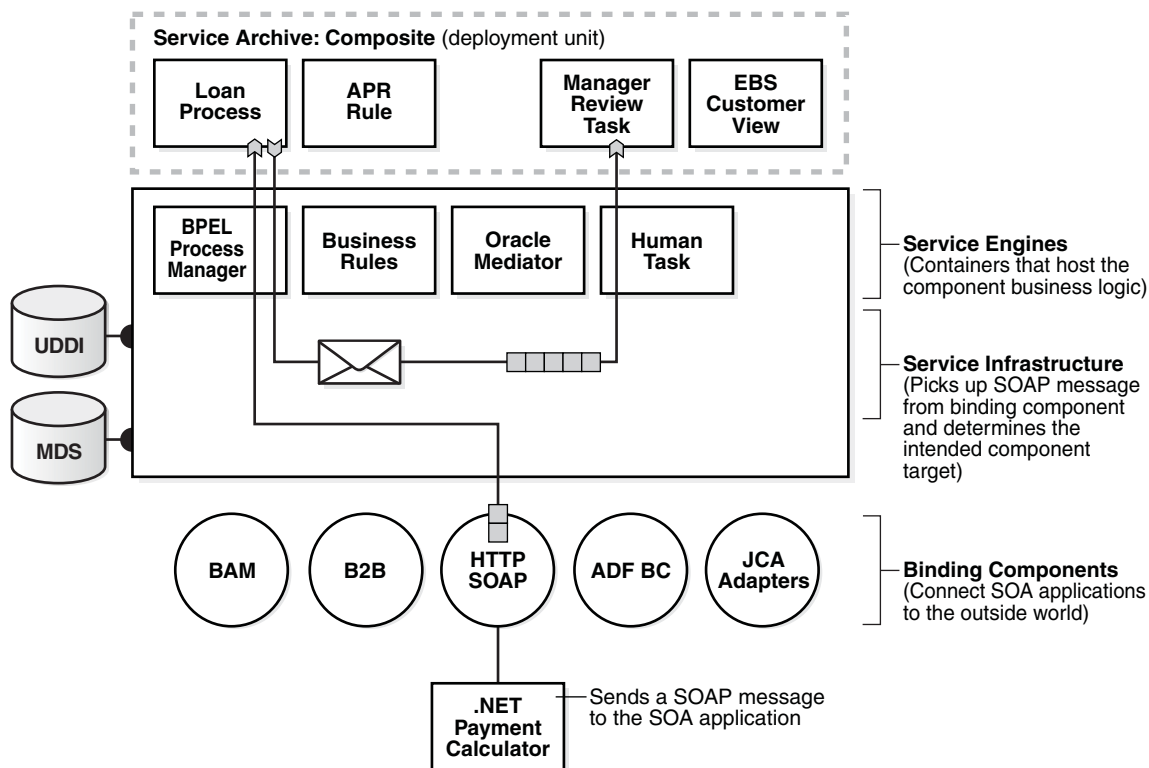
- Services to service components
- Service components to other service components
- Service components to references

1.6 Runtime Behavior of a SOA Composite Application

Figure 1–2 shows the operability of a SOA composite application using SCA technology. In this example, an external application (a .NET payment calculator) initiates contact with the SOA composite application.

For more information about descriptions of the tasks that services, references, service components, and wires perform in an application, see [Section 1.5, "Service Component Architecture within SOA Composite Applications."](#)

Figure 1–2 Runtime Behavior of SOA Composite Application



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

Table 1–2 describes the operability of the SOA composite application shown in Figure 1–1.

Table 1–2 Introduction to a SOA Composite Application Using SCA Technologies

Part	Description	Example of Use in Figure 1–1	See Section
Binding components	Establishes the connectivity between a SOA composite and the external world. There are two types: ▪ <i>Service</i> binding components provide an entry point to the SOA composite application. ▪ <i>Reference</i> binding components enable messages to be sent from the SOA composite application to external services.	The SOAP binding component <i>service</i>: ▪ 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. An example of a binding component <i>reference</i> in Figure 1–2 is the Loan Process application.	Section 1.5.1, "Service Components"
Service Infrastructure	Provides internal message transport	The Service Infrastructure: ▪ Receives the message from the SOAP binding component service. ▪ Posts the message for processing to the BPEL process service engine first and the human task service engine second.	Section 1.6.1, "Service Infrastructure"
Service engines (containers hosting service components)	Host the business logic or processing rules of the service components. Each service component has its own service engine.	The BPEL service engine: ▪ Receives the message from the Service Infrastructure for processing by the BPEL Loan Process application. ▪ Posts the message to the Service Infrastructure after completing the processing.	Section 1.6.2, "Service Engines"
UDDI and MDS	The MDS (Metadata Service) repository stores descriptions of available services. The UDDI advertises these services, and enables discovery and dynamic binding at runtime.	The SOAP service used in this composite application is stored in the MDS repository and can also be published to UDDI.	<i>Oracle Fusion Middleware Getting Started with Oracle SOA Suite</i>
SOA Archive: Composite (deployment unit)	The deployment unit that describes the composite application.	The SOA archive (SAR) of the composite application is deployed to the Service Infrastructure.	Section 1.6.3, "Deployed Service Archives"

1.6.1 Service Infrastructure

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

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

1.6.2 Service Engines

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.

There is a corresponding service engine of the same name for each service component. All service engines can interact in a single composite.

For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

1.6.3 Deployed Service Archives

The SAR is a SOA archive deployment unit. A SAR file is a special JAR file that requires a prefix of `sca_`. (for example, `sca_OrderBookingComposite_rev1.0.jar`). 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).

For more information, see [Chapter 40, "Deploying SOA Composite Applications."](#)

1.7 Approaches for Designing SOA Composite Applications

When creating a SOA composite application, you have a choice of approaches for building it:

- **Top-Down:** You analyze your business processes and identify activities in support of your process. When creating a composite, you define all the SOA components through the SOA Composite Editor. You create all the services first, and then build the BPEL process, referencing the created services.
- **Bottom-Up:** You analyze existing applications and assets to identify those that can be used as services. As you create a BPEL process, you build the services on an as-needed basis. This approach works well when IT must react to a change.

1.8 Learning Oracle SOA Suite

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

- **Getting Started:** *Oracle Fusion Middleware Getting Started with Oracle SOA Suite* introduces you to Oracle SOA Suite, its components, and provides you with a high-level understanding of what you can accomplish with the suite. Also, you can refer to the Oracle SOA Suite section of the Oracle Fusion Middleware 11g Release 1 documentation library for additional documentation.
- **Cue Cards in Oracle JDeveloper:** Oracle JDeveloper cue cards provide step-by-step support for the application development process using Oracle SOA Suite. They are designed to be used either with the included examples and a sample schema, or with your own data. Cue cards also include topics that provide more detailed background information, and viewlets that demonstrate how to complete the steps in the card. Cue cards provide a fast, easy way to become familiar with the basic features of Oracle SOA Suite, and to work through a simple end-to-end task. In Oracle JDeveloper, click **Help** > **Cue Cards** to access the cue cards.

- <https://soasamples.samplecode.oracle.com>: The SOA OTN provides access to various use case samples for Oracle SOA Suite and its components.

Note: While this guide primarily describes how to use Oracle SOA Suite with Oracle WebLogic Server, most of the information is also applicable to using Oracle SOA Suite with other third-party application servers. However, there may be some differences with using third-party application servers.

For information about these differences, see *Oracle Fusion Middleware Third-Party Application Server Guide*.

Developing SOA Composite Applications with Oracle SOA Suite

This chapter describes how to use Oracle JDeveloper to create a SOA composite application. This overview is intended to guide you through the basic steps of composite creation, along with describing key issues to be aware of when designing a composite application.

This chapter includes the following sections:

- [Section 2.1, "Creating a SOA Application"](#)
- [Section 2.2, "Adding Service Components"](#)
- [Section 2.3, "Adding Service Binding Components"](#)
- [Section 2.4, "Adding Reference Binding Components"](#)
- [Section 2.5, "Adding Wires"](#)
- [Section 2.6, "Adding Security"](#)
- [Section 2.7, "Deploying a SOA Composite Application"](#)
- [Section 2.8, "Managing and Testing a SOA Composite Application"](#)

2.1 Creating a SOA Application

The first steps in building a new application are to assign it a name and to specify the directory where to save source files. By creating an application using application templates provided by Oracle JDeveloper, you automatically get the organization of the workspace into projects, along with many of the configuration files required by the type of application you are creating.

2.1.1 How to Create a SOA Application and Project

You first create an application for the SOA project.

Note: In order to create and deploy SOA composite applications and projects, you must install the Oracle SOA Suite extension. For instructions on installing this extension for Oracle JDeveloper, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

To create an application:

1. Start Oracle JDeveloper Studio Edition Version 11.1.1.5.0.

2. If Oracle JDeveloper is running for the first time, specify the location for the Java JDK.
3. Create a new SOA composite application, as described in [Table 2–1](#).

Table 2–1 SOA Composite Application Creation

If Oracle JDeveloper...	Then...
Has no applications For example, you are opening Oracle JDeveloper for the first time.	In the Application Navigator in the upper left, click New Application .
Has existing applications	From the File main menu or the Application menu: 1. Select New > Applications. The New Gallery opens, where you can select different application components to create. 2. In the Categories tree, under the General node, select Applications. In the Items pane, select SOA Application and click OK.

The Create SOA Application wizard starts.

4. In the Name your application page, you can optionally change the name and location for your web project. If this is your first application, from **Application Template**, select **SOA Application**. Accept the defaults for the package prefix, and click **Next**.

Notes:

- Do *not* create an application name with spaces.
- Do *not* create applications and projects in directory paths that have spaces (for example, `c:\Program Files`).
- On a UNIX operating system, it is highly recommended to enable Unicode support by setting the `LANG` and `LC_All` environment variables to a locale with the UTF-8 character set. This action enables the operating system to process any character in Unicode. SOA technologies are based on Unicode. If the operating system is configured to use non-UTF-8 encoding, SOA components may function in an unexpected way. For example, a non-ASCII file name can make the file inaccessible and cause an error. Oracle does not support problems caused by operating system constraints.

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

5. In the Name your project page, you can optionally change the name and location for your SOA project. By default, Oracle JDeveloper adds the SOA project technology, the **composite.xml** that generates, and the necessary libraries to your model project. Click **Next**.

Note: Composite and component names cannot exceed 500 characters.

A project deployed to the same infrastructure *must* have a unique name across SOA composite applications. The uniqueness of a composite is determined by its project name. For example, do not perform the actions described in [Table 2–2](#). During deployment, the second deployed project (composite) overwrites the first deployed project (composite).

Table 2–2 Restrictions on Naming a SOA Project

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

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

6. In the Configure SOA Settings page, click **Empty Composite**, and click **Finish**.
7. From the **File** main menu, select **Save All**.

2.1.2 What Happens When You Create a SOA Application and Project

When you create a SOA application, Oracle JDeveloper creates a project that contains all the source files related to your application. Oracle JDeveloper automatically adds the following libraries needed for your SOA project:

- SOA Design time
- SOA Runtime
- BPEL Runtime
- Oracle Mediator Runtime
- MDS Runtime

You can then use Oracle JDeveloper to create additional projects needed for your application.

[Figure 2–1](#) shows the SOA Composite Editor for the **OrderBookingComposite** project contained within the **WebLogicFusionOrderDemo** application of the Fusion Order Demo.

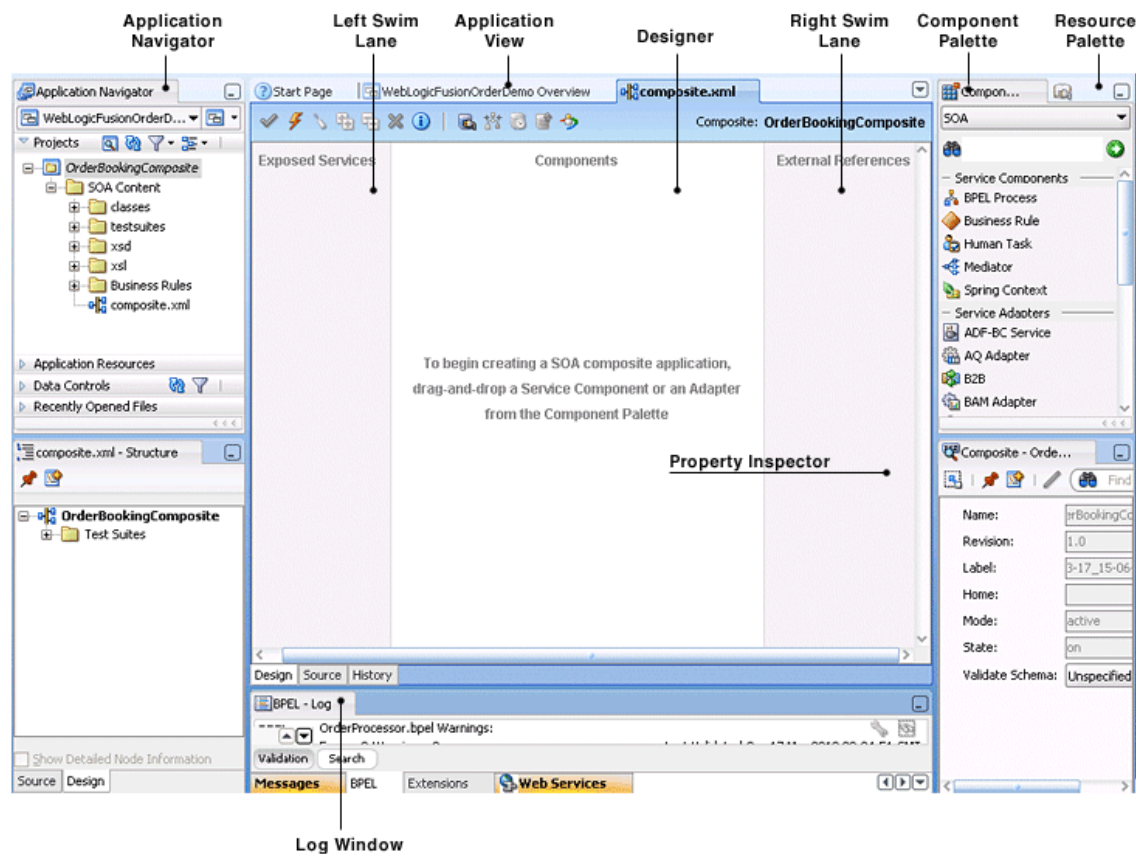
Figure 2–1 New Workspace for a SOA Composite Application

Table 2–3 describes the SOA Composite Editor.

Table 2–3 SOA Composite Editor

Element	Description
Application Navigator	Displays the key files for the specific service components included in the SOA project: - A composite.xml file that is automatically created when you create a SOA project. This file describes the entire composite assembly of services, service components, references, and wires. - The business rules service component file (<i>rules_name.decs</i>). Additional business rules files display under the Oracle > rules subfolder (<i>rules_name.rules</i>). - The Oracle Mediator service component file (<i>mediator_name.mplan</i>). - The BPEL process service component files (<i>process_name.bpel</i> and <i>process_name.wsdl</i>). - The human task service component file (<i>task_name.task</i>). - The spring service component file (<i>spring.xml</i>). - The componentType file that describes the services and references for each service component. This file ensures that the wiring you create between components works. - Additional subfolders for class files, XSDs (schemas), and XSLs (transformations).

Table 2–3 (Cont.) SOA Composite Editor

Element	Description
Designer	You drag service components, services, and references from the Component Palette into the composite in the designer. When you drag and drop a service component into the designer window, a corresponding property editor is invoked for performing configuration tasks related to that service component. For example, when you drag and drop the Oracle Mediator service component into the designer, the Mediator Editor is displayed that enables you to configure the Oracle Mediator service component. For all subsequent editing sessions, you double-click these service components to re-open their editors.
Left Swimlane (Exposed Services)	The left swimlane is for services, such as a web services or JCA adapters, providing an entry point to the SOA composite application.
Right Swimlane (External References)	The right swimlane is for references that send messages to external services in the outside world, such as web services and JCA adapters.
Component Palette	The component palette provides the various resources that you can use in a SOA composite. It contains the following service components and adapters: ■ Service components Displays the BPEL process, business rule, human task, Oracle Mediator, and spring components that can be dragged and dropped into the designer. ■ Service adapters Displays the JCA adapter (AQ, file, FTP, database, JMS, MQ, Oracle Applications, and socket), Oracle BAM binding component, B2B binding component, EJB binding component, ADF-BC binding component, direct binding component, HTTP binding component, and web service binding component that can be dragged into the left or right swimlanes. If the Component Palette does not display, select Component Palette from the View main menu.
Resource Palette	The Resource Palette provides a single dialog from which you can browse both local and remote resources. For example, you can access the following resources: ■ Shared local application metadata such as schemas, WSDLs, event definitions, business rules, and so on. ■ WSIL browser functionality that uses remote resources that can be accessed through an HTTP connection, file URL, or Application Server connection. ■ Remote resources that are registered in a Universal Description, Discover, and Integration (UDDI) registry. If the Resource Palette does not display, then select Resource Palette from the View main menu. You select these resources for the SOA composite application through the SOA Resource Browser dialog. This dialog is accessible through a variety of methods. For example, when you select the WSDL file to use with a service binding component or an Oracle Mediator service component or select the schema file to use in a BPEL process, the SOA Resource Browser dialog appears. Click Resource Palette at the top of this dialog to access available resources.

Table 2–3 (Cont.) SOA Composite Editor

Element	Description
Log Window	The Log window displays messages about application compilation, validation, and deployment.
Property Inspector	The Property Inspector displays properties for the selected service component, service, or reference. If the Property Inspector does not display, select Property Inspector from the View main menu.
Application View	The Application View shows the artifacts for the SOA composite application.

The **composite.xml** file displays as a tab in the designer and as a file in the Application Navigator. This file is automatically created when you create a new SOA project. This file describes the entire composite assembly of services, service components, and references. There is one **composite.xml** file for each SOA project.

When you work with the **composite.xml** file, you mostly use the designer, the Structure window, and the Property Inspector, as shown in [Figure 2–1](#). The designer enables you to view many of your files in a WYSIWYG environment, or you can view a file in an overview editor where you can declaratively make changes, or you can view the source code for the file. The Structure window shows the structure of the currently selected file. You can select objects in this window, and then edit the properties for the selection in the Property Inspector.

2.1.3 What You May Need to Know About Opening the composite.xml File Through a SOA-MDS Connection

If you create a SOA-MDS connection in Oracle JDeveloper, expand the connection, and attempt to open the **composite.xml** file of a composite from the Resource Palette, the file may not load correctly. Only open a composite from the Application Navigator.

For information about the Oracle Metadata Services (MDS) repository, see *Oracle Fusion Middleware Administrator's Guide*.

2.2 Adding Service Components

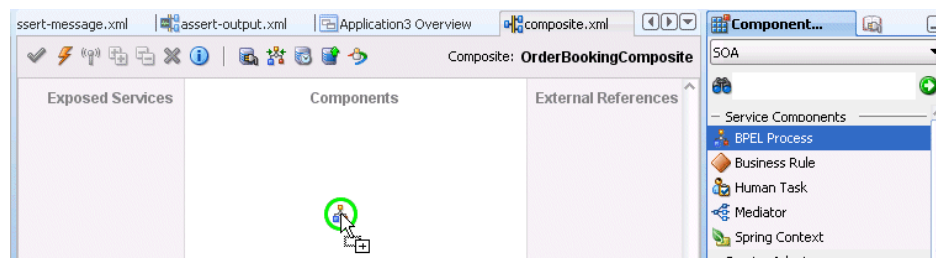
Once you create your application, often the next step is to add service components that implement the business logic or processing rules of your application. You can use the Component Palette from the SOA Composite Editor to drag and drop service components into the composite.

2.2.1 How to Add a Service Component

To add a service component:

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

[Figure 2–2](#) shows a BPEL process being added to the designer.

Figure 2–2 Adding BPEL Process to Composite

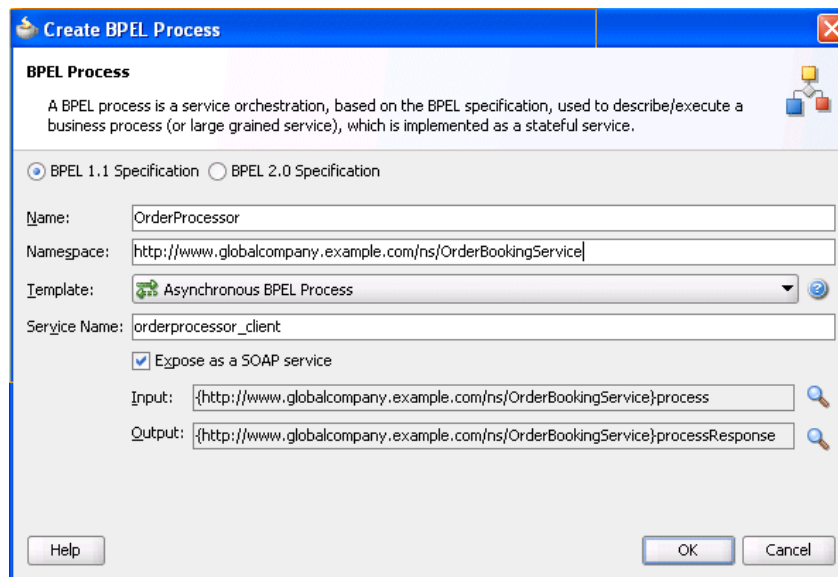
A specific dialog for the selected service component is displayed. [Table 2–4](#) describes the available editors.

Table 2–4 Starting Service Component Editors

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

3. Configure the settings for a service component. For help with a service component dialog, click **Help** or press **F1**. Click **Finish**.

[Figure 2–3](#) shows the BPEL Process dialog with data entered to create the **OrderProcessor** BPEL process for the **WebLogicFusionOrderDemo** application of the Fusion Order Demo. The process is selected to be asynchronous. The **Expose as a SOAP Service** option directs Oracle JDeveloper to create this service component automatically connected to an inbound web service.

Figure 2–3 Create BPEL Process Dialog


Create BPEL Process

A BPEL process is a service orchestration, based on the BPEL specification, used to describe/execute a business process (or large grained service), which is implemented as a stateful service.

☒ BPEL 1.1 Specification ☐ BPEL 2.0 Specification

Name:

Namespace:

Template:

Service Name:

☒ Expose as a SOAP service

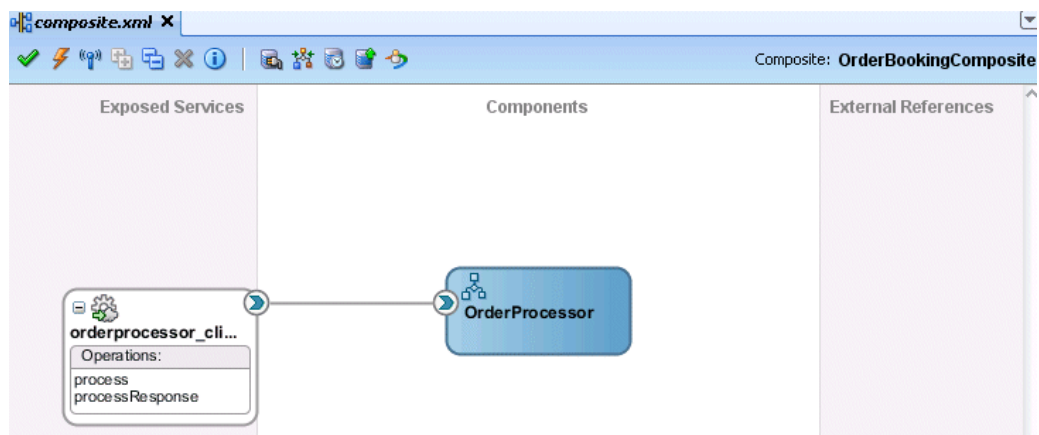
Input:

Output:

Buttons: Help, OK, Cancel

4. Click **OK**.

The service component displays in the designer. [Figure 2–4](#) shows the **OrderProcessor** BPEL process added to the **composite.xml** file. A SOAP service binding component called **orderprocessor_client_ep** in the left swimlane provides the outside world with an entry point into the SOA composite application. If the **Expose as a SOAP Service** option was not selected in the Create BPEL Process dialog, the **orderprocessor_client_ep** service would not display. [Section 2.3.1, "How to Add a Service Binding Component,"](#) describes how you later add a service.

Figure 2–4 BPEL Process in Composite

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

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

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

Note the following details about adding service components:

- Create a service component from either the SOA Composite Editor or the designer of another component. For example, you can create a human task component from the SOA Composite Editor or the Oracle BPEL Designer.
- Use the Resource Palette to browse for service components defined in the SOA Composite Editor, and those deployed.

Note the following details about deleting service components:

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

2.2.3 How to Edit a Service Component

You modify a service component to define specific details about the service component.

To edit a service component:

1. Double-click the service component in the designer to display the appropriate editor or designer, as described in [Table 2–5](#).

Table 2–5 Starting SOA Service Component Wizards and Dialogs

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

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 designer.

For help with a service component editor, click **Help** or press **F1**.

2. Click **Finish**.
3. Modify the settings for a service component. For help with a service component editor or designer, click **Help** or press **F1**.
4. Click **Finish**.
5. In the Application Navigator, double-click **composite.xml** or single-click **composite.xml** above the designer.

This action returns you to the SOA Composite Editor.

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

2.3 Adding Service Binding Components

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

2.3.1 How to Add a Service Binding Component

Notes:

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

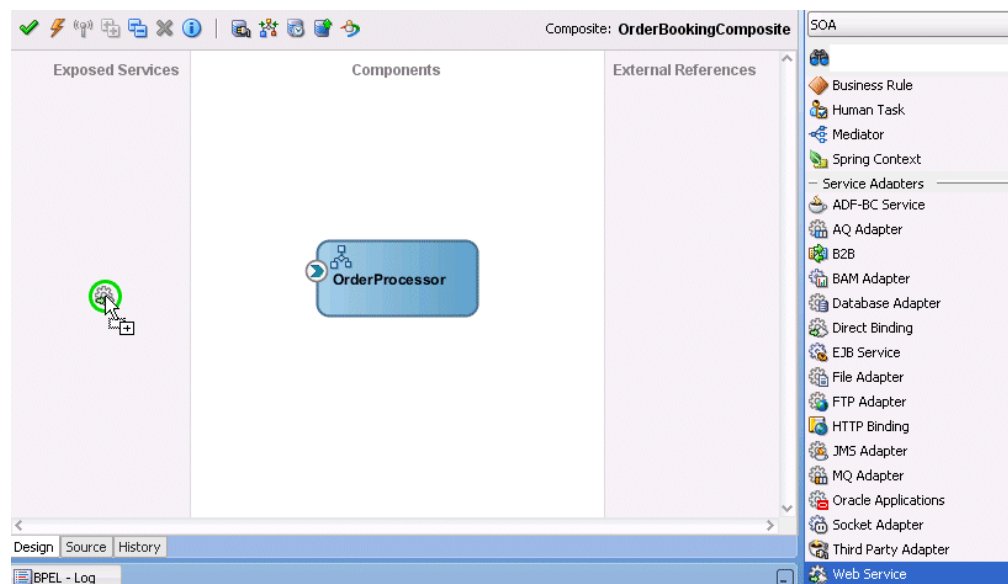
You can use the Component Palette from the SOA Composite Editor to drag and drop service binding components to the composite.

To add a service binding component:

1. From the Component Palette, select **SOA**.
2. From the **Service Adapters** list, drag a service to the *left* swimlane to define the service interface.

Figure 2–5 shows a web service being added to the designer.

Figure 2–5 Adding Web Service to Composite



A specific dialog for the selected service displays. [Table 2–6](#) describes the available editors.

Table 2–6 Service Editors

Dragging This Service...	Invokes The...
Web service	Create Web Service dialog to create a web invocation service.
Adapters	Adapter Configuration Wizard to guide you through integration of the service with database tables, database queues, file systems, FTP servers, Java Message Services (JMS), IBM WebSphere MQ, BAM servers, sockets, or Oracle E-Business Suite applications.
ADF-BC Service	Create ADF-BC Service dialog to create a service data object (SDO) invocation service.
B2B	B2B Wizard to guide you through selection of a document definition.
EJB Service	Create EJB Service to create an Enterprise JavaBeans service for using SDO parameters or Java interfaces with Enterprise JavaBeans.
HTTP Binding	Create HTTP Binding Wizard to create HTTP binding. This wizard enables you to invoke SOA composite applications through HTTP POST and GET operations.
Direct Binding	Create Direct Binding Service dialog to invoke a SOA composite application and exchange messages over a remote method invocation (RMI) in the inbound direction.

3. Configure the settings for the service. For help with a service editor, click **Help** or press **F1**.
4. Click **Finish**.

[Figure 2–6](#) shows the Web Service dialog with data entered to create the `orderprocessor_client_ep` service for the **OrderProcessor** BPEL process.

Figure 2–6 Create Web Service Dialog

Web Service

Create a web service for services external to the SOA composite.

Name:

Type:

WSDL URL:

Port Type:

Callback Port Type:

☐ copy wsdl and its dependent artifacts into the project.

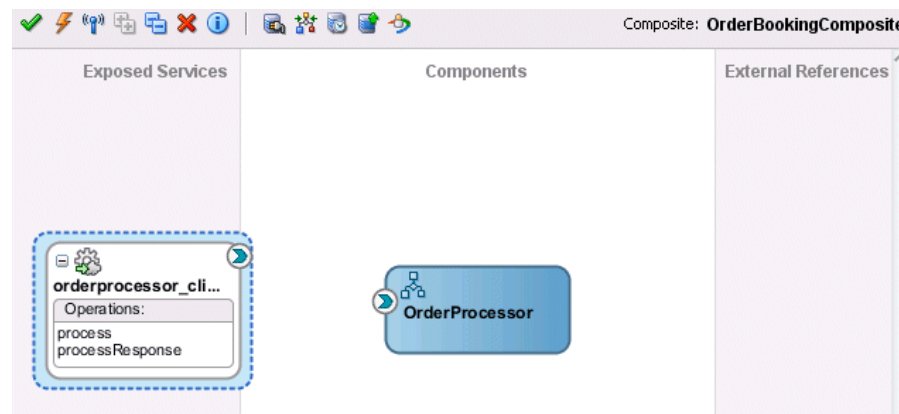
Note: Keeping a copy of a WSDL may result in synchronization issues if the remote WSDL is updated. It is recommended not make local copies - this should be reserved for situations such as offline designing.

Transaction Participation:

Version:

5. Click **OK**.

The service binding component displays in the left swimlane. [Figure 2–7](#) shows the `orderprocessor_client_ep` service binding component added to the `composite.xml` file.

Figure 2–7 Web Service in Composite

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

2.3.2 How to Add a WSDL for a Web Service

As described in [Section 2.3.1, "How to Add a Service Binding Component,"](#) a web service is a type of binding component that you can add to a SOA composite application. You must select the WSDL file for the web service.

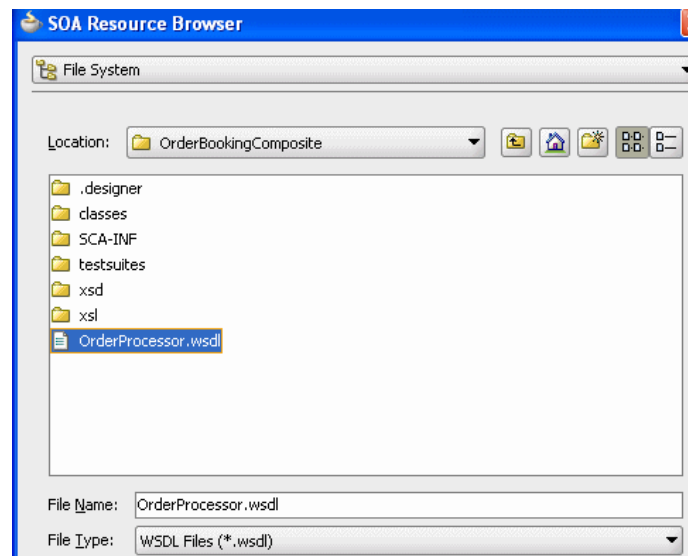
To add a WSDL for a web service:

1. In the Component Palette, select **SOA**.
2. From the **Service Adapters** list, drag a **Web Service** to the *left* swimlane.
This invokes the Create Web Service dialog shown in [Figure 2–6](#).
3. Enter the details shown in [Table 2–7](#):

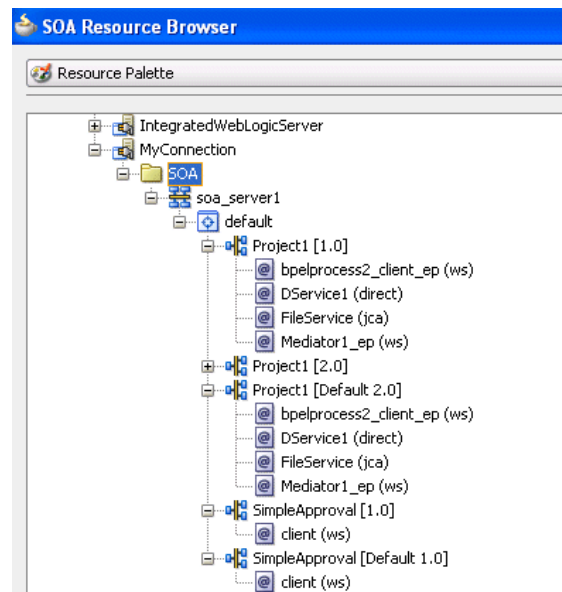
Table 2–7 Create Web Service Dialog Fields and Values

Field	Value
Name	Enter a name for the service.
Type	Select the type (message direction) for the web service. Since you dragged the web service to the left swimlane, the Service type is the correct selection, and displays by default: ■ 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 service in the outside world Since this example describes how to create an entry point to the SOA composite application, Service is selected.

4. Select the WSDL file for the service. There are three methods for selection:
 - a. To the right of the **WSDL URL** field, click the first icon and select an existing WSDL file from the local file system (for this example, **OrderProcessor.wsdl** is selected). Note that **File System** in the list at the top of the dialog is automatically selected. [Figure 2–8](#) provides details.

Figure 2–8 WSDL File Selection

- b. To the right of the **WSDL URL** field, click the first icon and select **Resource Palette** from the list at the top of the dialog, as shown in Figure 2–9. This action enables you to use existing WSDL files from other applications.

Figure 2–9 Use of Existing WSDL files from Other Applications

- c. To the right of the **WSDL URL** field, click the second icon to automatically generate a WSDL file from a schema. Figure 2–10 shows the Create WSDL dialog. Default values for the WSDL file name, directory location, namespace, port type, operation name, and interface type are displayed. If the specified directory is not the subdirectory of the current project, a warning message is displayed. If the specified directory does not exist, it is automatically created. You can modify the default values.

Figure 2–10 Automatic Generation of WSDL File

Create WSDL

File Name:

Directory:

Namespace:

Port Type:

Operation:

Interface Type:

Input:

Message Part Name	Element or Type	Schema URL

☐ Generate partnerlinkType extension

Help OK Cancel

- Click the **Add** icon above the **Input** table to display the Add Message Part dialog to add a new WSDL message part. If the WSDL file contains multiple messages, you can add a message part for each one. You can select XML schema simple types, project schema files, and project WSDL files for a message part.

For more information, click **Help**.

- Click **OK** to return to the Create Web Service dialog.
- Note the additional details described in [Table 2–8](#):

Table 2–8 Create Web Service Dialog Fields and Values

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

- Click **OK**.
- From the **File** main menu, select **Save All**.

Notes:

- Do *not* manually update the WSDL location in the WSDL file in **Source View**. This action is not supported. Only updates made in **Design View** are supported.
- WSDL namespaces must be unique. Do not just copy and rename a WSDL. Ensure that you also change the namespaces.

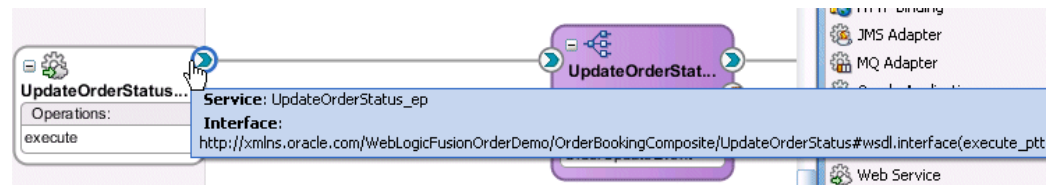
2.3.3 How to View Schemas

You can view all schemas used by the interface's WSDL file and, if you want, choose a new message schema for a selected message part in the Update Interface dialog.

To view schemas:

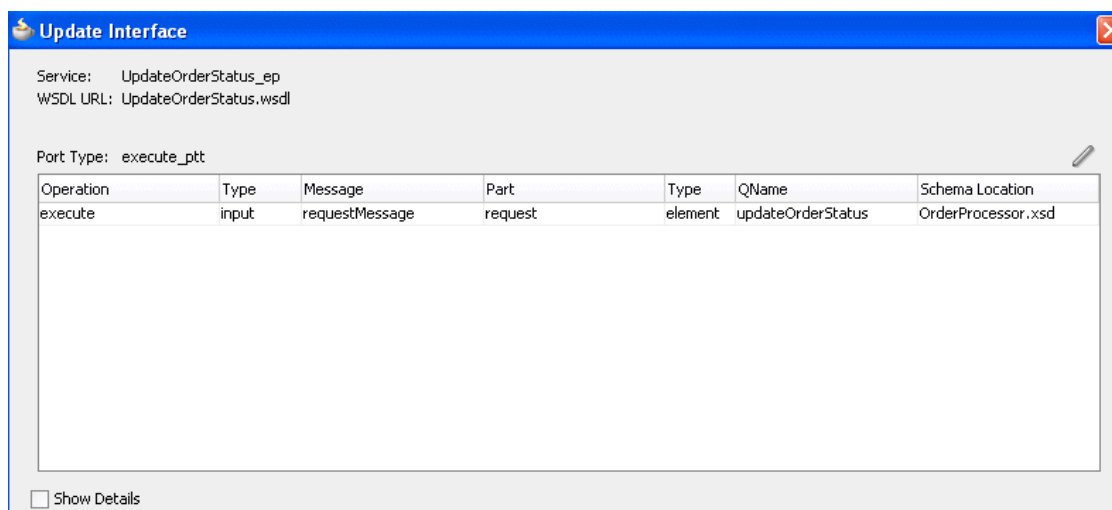
1. Double-click the small arrow handle that appears on the specific binding component or service component. [Figure 2–11](#) provides details.

Figure 2–11 Selection of Inbound Interface Handle



The Update Interface dialog shown in [Figure 2–12](#) displays all schemas currently used by the WSDL file.

Figure 2–12 Update Interface Dialog



2. If you want to select a new message schema, click **Help** or press **F1** for instructions.

2.3.4 How to Edit a Service Binding Component

After initially creating a service, you can edit its contents at a later time. Double-click the component icon to display its appropriate editor or wizard. [Table 2–9](#) provides an overview.

Table 2–9 Starting Service Wizards and Dialogs

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

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

Note the following detail about adding services:

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

Note the following detail about deleting services:

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

2.4 Adding Reference Binding Components

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

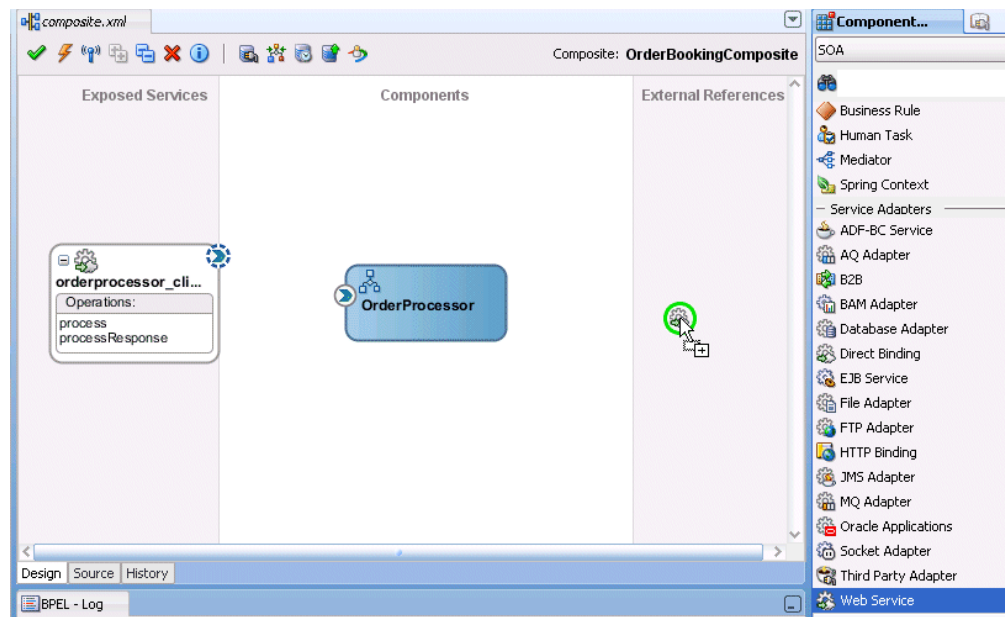
2.4.1 How to Add a Reference Binding Component

You can use the Component Palette from the SOA Composite Editor to drag and drop reference binding components to the composite.

To add a reference binding component:

1. From the Component Palette, select **SOA**.
2. From the **Service Adapters** list, drag a service to the *right* swimlane.

[Figure 2–13](#) shows a web service being added to the designer.

Figure 2–13 Adding Web Service to Composite

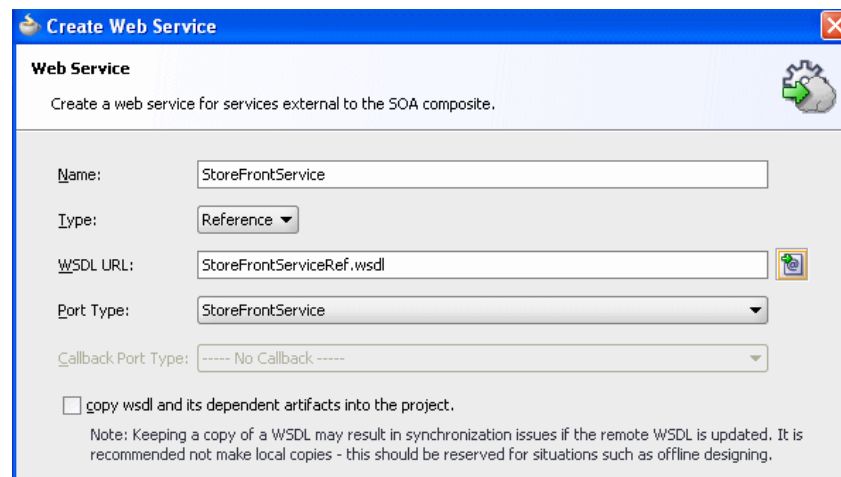
A specific dialog or wizard for the selected reference displays. [Table 2–10](#) describes the available editors.

Table 2–10 Reference Editors

Dragging This Service...	Invokes The...
Web Service	Create Web Service dialog to create a web invocation service.
Adapters	Adapter Configuration Wizard to guide you through integration of the service with database tables, database queues, file systems, FTP servers, Java Message Services (JMS), IBM WebSphere MQ, BAM servers, sockets, or Oracle E-Business Suite applications.
ADF-BC Service	Create ADF-BC Service dialog to create a service data object (SDO) invocation service.
B2B	B2B Wizard to guide you through selection of a document definition.
EJB Service	Create EJB Service dialog to create an Enterprise JavaBeans service for using SDO parameters with Enterprise JavaBeans.
HTTP Binding	Create HTTP Binding Wizard to create HTTP binding. This wizard enables you to invoke SOA composite applications through HTTP POST and GET operations, and invoke HTTP endpoints through HTTP POST and GET operations.
Direct Binding	Create Direct Binding Service Dialog to invoke an Oracle Service Bus (OSB) flow or another SOA composite application.

3. Configure the settings for the reference binding component. For help with a reference editor, click **Help** or press **F1**.
4. Click **Finish**.

[Figure 2–14](#) shows the Create Web Service dialog with data entered to create a reference.

Figure 2–14 Create Web Service Dialog


Create Web Service

Create a web service for services external to the SOA composite.

Name:

Type:

WSDL URL:

Port Type:

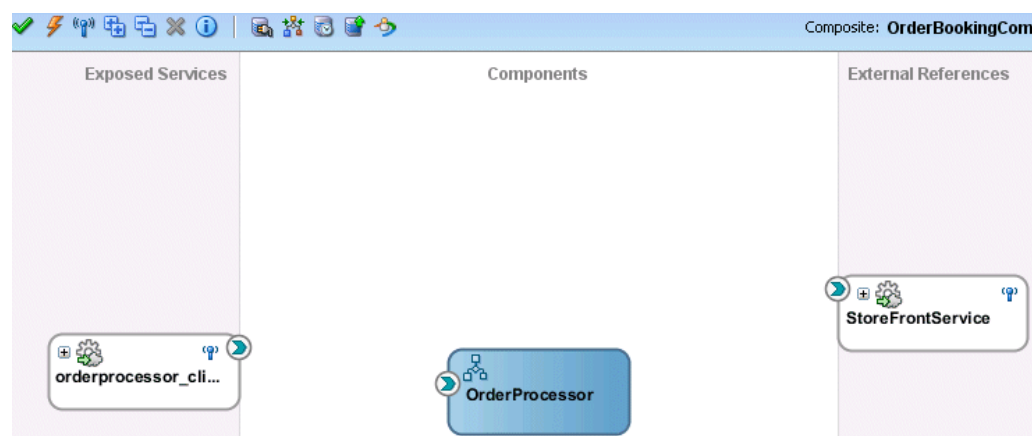
Callback Port Type:

☐ copy wsdl and its dependent artifacts into the project.

Note: Keeping a copy of a WSDL may result in synchronization issues if the remote WSDL is updated. It is recommended not make local copies - this should be reserved for situations such as offline designing.

5. Click **OK**.

The reference binding component displays in the right swimlane. [Figure 2–15](#) shows the **StoreFrontService** reference added to the SOA composite application.

Figure 2–15 Web Service in Composite

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

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

Note the following detail about adding references:

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

Note the following details about deleting references:

- When a reference for a service component is deleted, the associated wire is also deleted and the service component is notified so it can update its metadata. For

example, when a reference is deleted from a BPEL service component, the service component is notified to delete the partner link in its BPEL metadata.

- Deleting a reference connected to a wire clears the reference and the wire.

2.4.3 What You May Need to Know About WSDL References

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

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

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

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

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

2.4.4 What You May Need to Know About Mixed Message Types in a WSDL File

If a BPEL process has multiple WSDL messages declared in its WSDL file and one or more messages have their parts defined to be of some type, whereas other messages have their parts defined to be of some element, runtime behavior can become unpredictable. This is because these WSDLs are considered to have mixed type messages. For example, assume there are multiple copy actions within an assign activity. These copy actions attempt to populate an output variable that has multiple parts:

- Part 1 is declared as an `xsd:string` type.
- Part 2 is declared as an `xsd:int` type.
- Part 3 is declared as an element of a custom-designed complex type.

This behavior is not supported.

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

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

Select the default WSDL to use in the **Resource Palette** in Oracle JDeveloper.

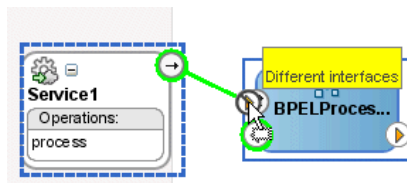
1. In the Create Web Service dialog, click the icon to the right of the **WSDL URL** field to invoke the SOA Resource Browser dialog.
2. Select **Resource Palette** from the list at the top.
3. Expand the nodes under the **Application Server** connection or **WSIL** connection to list all deployed composites and revisions. The default revision is identified by the word **Default** in the title. For example, **OrderBookingComposite [Default]**.
4. Select the appropriate default endpoint and click **OK**.

2.5 Adding Wires

You wire (connect) services, service components, and references. For this example, you wire the web service and service component. Note the following:

- Since a 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, an Oracle Mediator can only have one inbound service. Therefore, it rejects attempts to create a second service.
 - You cannot drag an outbound service (external reference) to a business rule, because business rules do not support references. When dragging a wire, the user interface highlights the interfaces that are valid targets.
- You cannot wire services and composites that have different interfaces. For example, you cannot connect a web service configured with a synchronous WSDL file to an asynchronous BPEL process. [Figure 2–16](#) provides details.

Figure 2–16 *Limitations on Wiring Services and Composites with Different Interfaces*



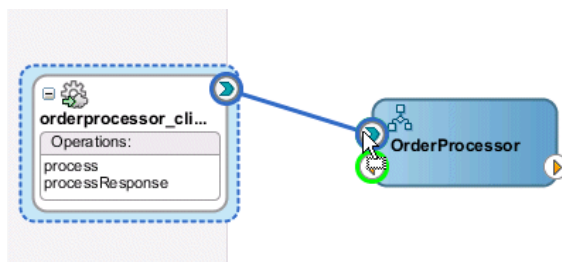
The service and reference must match, meaning the interface and the callback must be the same. If you have two services that have different interfaces, you can place an Oracle Mediator between the two services and perform a transformation between the interfaces.

2.5.1 How to Wire a Service and a Service Component

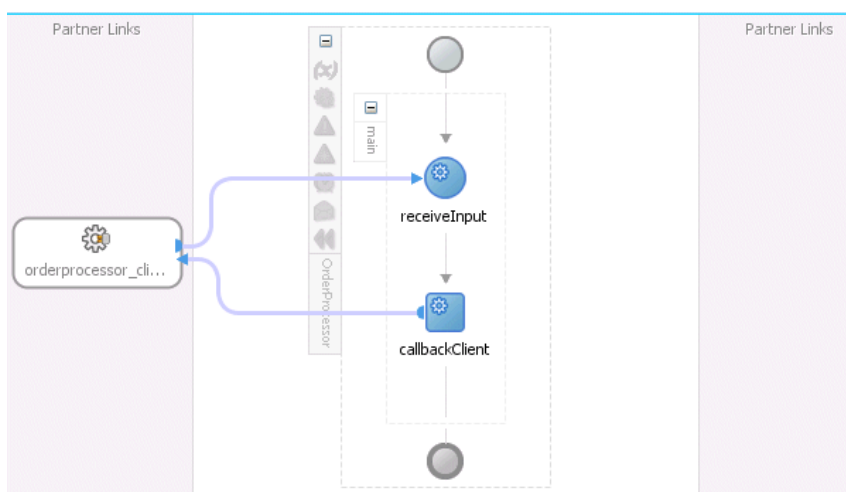
You can wire a service binding component to a service component from the SOA Composite Editor.

To wire a service and a service component:

1. From a service reference handle, drag a wire to the service component interface, as shown in [Figure 2–17](#).

Figure 2–17 Wire Connection

2. If the service component is a BPEL process, double-click the BPEL process and note that the service displays as a partner link in the left swimlane, as shown in [Figure 2–18](#).

Figure 2–18 Display of the Service as a Partner Link in the BPEL Process

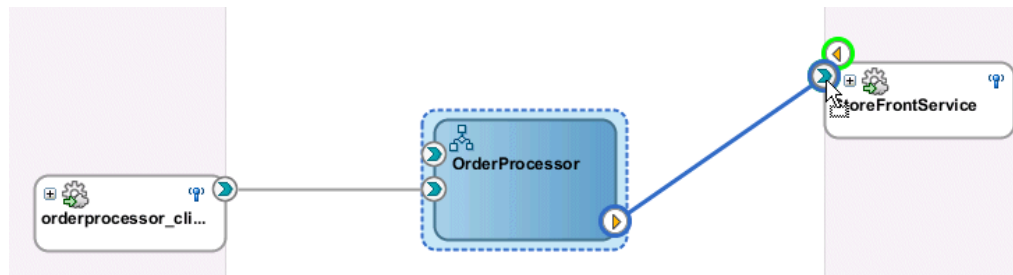
3. Select **Save All** from the **File** main menu.

2.5.2 How to Wire a Service Component and a Reference

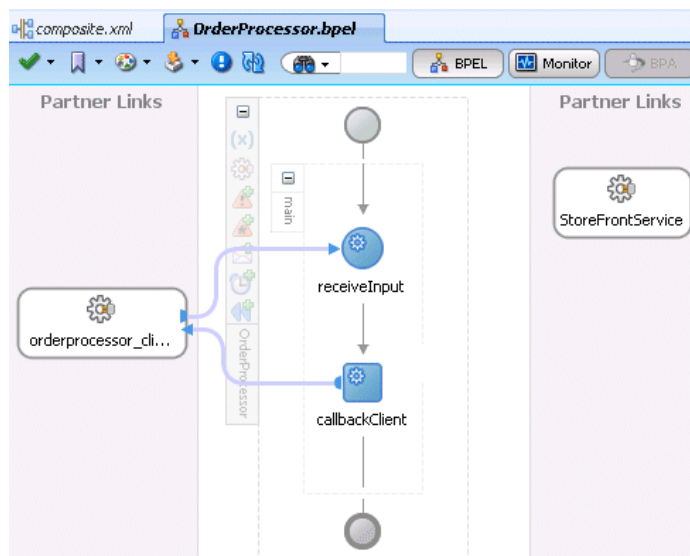
You can wire a service component to a reference binding component from the SOA Composite Editor.

To wire a service component and a reference:

1. In the Application Navigator, double-click **composite.xml** or single-click **composite.xml** above the designer.
2. From the service component, drag a wire to the reference, as shown in [Figure 2–19](#).

Figure 2–19 Wiring of a Service Component and Reference

3. If the service component is a BPEL process, double-click the BPEL process and note that the reference displays as a partner link in the right swimlane, as shown in Figure 2–20.

Figure 2–20 Display of the Reference as a Partner Link in the BPEL Process

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

The `orderprocessor_client_ep` service binding component shown in Example 2–1 provides the entry point to the composite.

Example 2–1 Service

```
<service name="orderprocessor_client_ep"
  ui:wsdlLocation="oramds:/apps/FusionOrderDemoShared
/services/orderbooking/OrderBookingProcessor.wsdl">
  <interface.wsdl interface="http://www.globalcompany.example.com/ns
/OrderBookingService#wsdl.interface(OrderProcessor)"
  <binding.adf serviceName="OrderProcessorService" registryName="/">
    <callback>
      <binding.ws port="http://www.globalcompany.example.com/ns
/OrderBookingService#wsdl.endpoint(orderprocessor_client_ep/OrderProcessorCallback_
pt)"/>
    </callback>
  </binding.adf>
</interface.wsdl>
</service>
```


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

Example 2-2 Service Component

```
<component name="OrderProcessor">
  <implementation.bpel src="OrderProcessor.bpel"/>
</component>
```

A reference binding component named StoreFrontService is shown in [Example 2-3](#). The reference provides access to the external service in the outside world.

Example 2-3 Reference

```
<reference name="StoreFrontService"
  ui:wsdlLocation="oramds:/apps/FusionOrderDemoShared
/services/oracle/fodemo/storefront/store/service/common/serviceinterface/StoreFrontService.wsdl">
  <interface.wsdl
    interface="www.globalcompany.example.com#wsdl.interface(StoreFrontService)"/>
  <binding.ws
    port="www.globalcompany.example.com#wsdl.endpoint(StoreFrontService/StoreFrontServiceSoapHttpPort)"/>

  location="oramds:/apps/FusionOrderDemoShared/services/oracle/fodemo/storefront/store/service/common/serviceinterface/StoreFrontService.wsdl"/>
</reference>
```

In [Example 2-4](#), the communication (or wiring) between service components is described:

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

Example 2-4 Wires

```
<wire>
  <source.uri>orderprocessor_client_ep</source.uri>
  <target.uri>OrderProcessor/orderprocessor_client_ep</target.uri>
</wire>

<wire>
  <source.uri>OrderProcessor/StoreFrontService</source.uri>
  <target.uri>StoreFrontService</target.uri>
</wire>
```

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

Note the following details about adding wires:

- A service component can be wired to another service component if its reference matches the service of the target service component. Note that the match implies the same interface and callback interface.

- Adding the following wiring between two Oracle Mediator service components causes an infinite loop:
 - Create a business event.
 - Create an Oracle Mediator service component and subscribe to the event.
 - Create a second Oracle Mediator service component to publish the same event.
 - Wire the first Oracle Mediator to the second Oracle Mediator component service.

If you remove the wire between the two Oracle Mediators, then for every message, the second Oracle Mediator can publish the event and the first Oracle Mediator can subscribe to it.

Note the following details about deleting wires:

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

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

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

See [Section 2.3.3, "How to View Schemas"](#) for details about the Update Interface dialog.

2.6 Adding Security

As you create your SOA composite application, you can secure web services by attaching policies to service binding components, service components, and reference binding components. For more information about implementing policies, see [Chapter 39, "Enabling Security with Policies."](#)

2.7 Deploying a SOA Composite Application

Deploying the SOA composite application involves creating a connection to an Oracle WebLogic Server and deploying an archive of the SOA composite application to an Oracle WebLogic Server managed server. For more information about deploying SOA composite applications, see [Chapter 40, "Deploying SOA Composite Applications."](#)

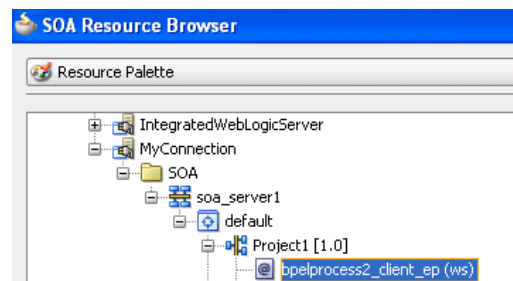
2.7.1 How to Invoke Deployed Composites

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

To invoke other composites:

1. Create a web service or partner link through one of the following methods.
 - a. In the SOA Composite Editor, drag a **Web Service** from the Component Palette to the **External References** swimlane.
 - b. In Oracle BPEL Designer, drag a **Partner Link** from the Component Palette to the *right* swimlane.
2. Access the SOA Resource Browser dialog based on the type of service you created.
 - a. For the Create Web Service dialog, click the **Find existing WSDLs** icon.
 - b. For the Edit Partner Link dialog, click the **SOA Resource Browser** icon.
3. From the list at the top, select **Resource Palette**.
4. Expand the tree to display the application server connection to the Oracle WebLogic Administration Server on which the SOA composite application is deployed.
5. Expand the application server connection.
6. Expand the **SOA** folder. [Figure 2–21](#) provides details.

Figure 2–21 Browse for a SOA Composite Application



7. Select the composite service.
8. Click OK.

2.8 Managing and Testing a SOA Composite Application

As you build and deploy a SOA composite application, you manage and test it using a combination of Oracle JDeveloper and Oracle Enterprise Manager Fusion Middleware Control.

2.8.1 How to Manage Deployed Composites

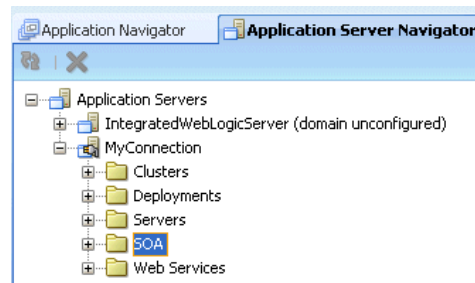
You can manage deployed SOA composite applications from the Application Server Navigator in Oracle JDeveloper. Management tasks consist of undeploying, activating, retiring, turning on, and turning off SOA composite application revisions.

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

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

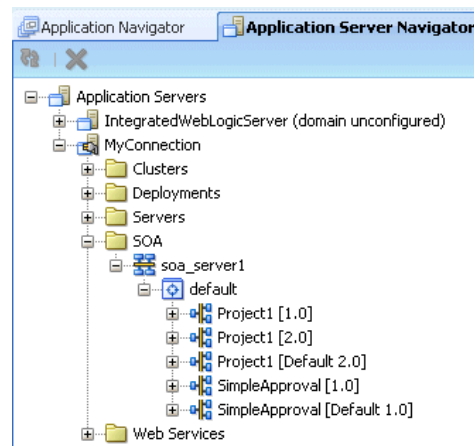
The **SOA** folder appears, as shown in [Figure 2–22](#). The **SOA** folder displays all deployed SOA composite application revisions and services. You can browse all applications deployed on all Oracle WebLogic Administration Servers, managed Oracle WebLogic Servers, and clustered Oracle WebLogic Servers in the same domain. [Figure 2–22](#) provides details.

Figure 2–22 Application Server Navigator



3. Expand the **SOA** folder.
4. Expand the partition in which the composite application is deployed.

Deployed SOA composite applications and services appear, as shown in [Figure 2–23](#).

Figure 2–23 Deployed SOA Composite Applications

5. Right-click a deployed SOA composite application.
6. Select an option to perform. The options that display for selection are based upon the current state of the application. [Table 2–11](#) provides details.

Table 2–11 SOA Composite Application Options

Option	Description
Stop	Shuts down a running SOA composite application revision. Any request (initiating or a callback) to the composite is rejected if the composite is shut down. Note: The behavior differs based on which binding component is used. For example, if it is a web service request, it is rejected back to the caller. A JCA adapter binding component may do something else in this case (for example, put the request in a rejected table). This option displays when the composite application has been started.
Start	Restarts a composite application revision that was shut down. This action enables new requests to be processed (and not be rejected). No recovery of messages occurs. This option displays when the composite application has been stopped.
Retire	Retires the selected composite revision. If the process life cycle is retired, you cannot create a new instance. Existing instances are allowed to complete normally. An initiating request to the composite application is rejected back to the client. The behavior of different binding components during rejection is the same as with the shut down option. A callback to an initiated composite application instance is delivered properly. This option displays when the composite application is active.
Activate	Activates the retired composite application revision. Note the following behavior with this option: ■ All composite applications are automatically active when deployed. ■ Other revisions of a newly deployed composite application remain active (that is, they are not automatically retired). If you want, you must explicitly retire them. This option displays when the application is retired.

Table 2–11 (Cont.) SOA Composite Application Options

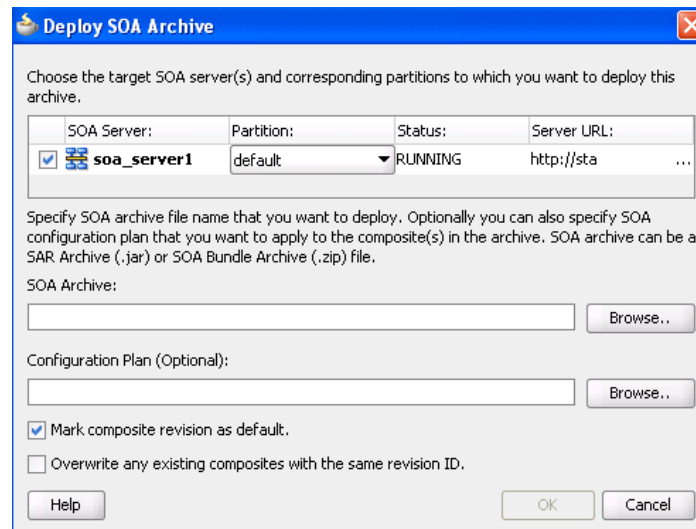
Option	Description
Undeploy	Undeploys the selected composite application revision. The consequences of this action are as follows: ■ You can no longer configure and monitor this revision of the composite application. ■ You can no longer process instances of this revision of the composite application. ■ You cannot view previously completed processes. ■ The state of currently running instances is changed to stale and no new messages sent to this composite are processed. ■ If you undeploy the default revision of the composite application (for example, 2.0), the next available revision of the composite application becomes the default (for example, 1.0).
Set Default Revision	Sets the selected composite application revision to be the default.

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

You are prompted to select the following:

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

Figure 2–24 provides details.

Figure 2–24 Deploy SOA Archive Dialog

For more information, see the following documentation:

- [Chapter 40, "Deploying SOA Composite Applications"](#) for details about creating a deployment profile and a configuration plan and deploying an existing SOA archive
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for details about managing deployed SOA composite applications from Oracle Enterprise Manager Fusion Middleware Control.

2.8.2 How to Test a Deployed Composite

After you deploy a SOA composite application, you can initiate a test instance of it from the Test Web Service page in Oracle Enterprise Manager Fusion Middleware Control to verify the XML payload data. For more information about initiating a test instance, see the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

In addition to creating a test instance, you can also simulate the interaction between a SOA composite application and its web service partners before deployment in a production environment. This helps to ensure that a process interacts with web service partners as expected by the time it is ready for deployment to a production environment. For more information about creating a unit test, see [Chapter 41, "Automating Testing of SOA Composite Applications."](#)

Introduction to the SOA Sample Application

This chapter introduces the SOA sample application that can be used with this guide. The WebLogic Fusion Order Demo application of the Fusion Order Demo demonstrates various capabilities of Oracle SOA Suite and is used as an example throughout this guide.

This chapter includes the following sections:

- [Section 3.1, "Introduction to the Fusion Order Demo"](#)
- [Section 3.2, "Setting Up the Fusion Order Demo Application"](#)
- [Section 3.3, "Taking a Look at the WebLogic Fusion Order Demo Application"](#)
- [Section 3.4, "Understanding the OrderBookingComposite Flow"](#)
- [Section 3.5, "Deploying Fusion Order Demo"](#)
- [Section 3.6, "Running Fusion Order Demo"](#)
- [Section 3.7, "Viewing Data Sent to Oracle BAM Server"](#)
- [Section 3.8, "Undeploying the Composites for the WebLogic Fusion Order Demo Application"](#)

3.1 Introduction to the Fusion Order Demo

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

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

3.1.1 Store Front Module

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

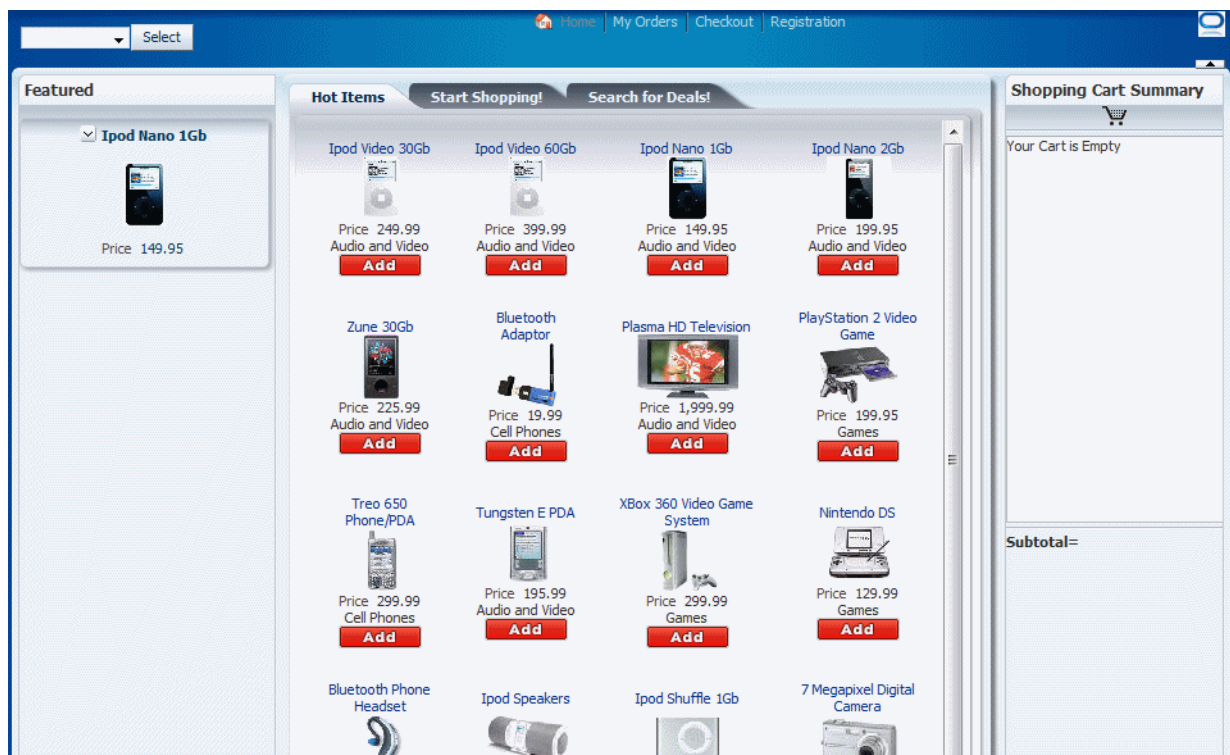
The Store Front module sells electronic devices through a storefront-type web application.

The Store Front module contains the following projects:

- **StoreFrontService:** This project provides access to the storefront data and provides transaction support to update data for customers, orders, and products.
- **StoreFrontUI:** This project provides web pages that the customer uses to browse the storefront, place orders, register on the site, view order information, and update the user profile.

Figure 3–1 shows the Home page of the Store Front module user interface. It shows the featured products that the site wants to promote and provides access to the full catalog of items. Products are presented as images along with the name of the product. Page regions divide the product catalog area from other features that the site offers.

Figure 3–1 StoreFrontUI Home Page



From the home page, you can browse the web site as an anonymous user, then log in as a registered customer to place an order.

The Fusion Order Demo application ships with predefined customer data. Because the Fusion Order Demo application implements Oracle ADF security to manage access to Oracle ADF resources, only the authenticated user can view orders in their cart.

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

3.1.2 WebLogic Fusion Order Demo Application

The WebLogic Fusion Order Demo application processes orders placed in the Store Front module. It uses the following Oracle SOA Suite components:

- Oracle Mediator
- Oracle BPEL process
- Human workflow (using a human task)

- Oracle Business Rules
- Spring
- Oracle User Messaging Service
- Oracle Business Activity Monitoring
- Oracle Metadata Repository

Once an order has been placed by using the Store Front module, the WebLogic Fusion Order Demo application processes the order. When processing an order, it uses various internal and external applications, including a customer service application, a credit validation system, and both an internal vendor and external vendor. For example, the internal vendor (InternalWarehouseService) and external vendor (ExternalPartnerSupplier), are sent information for every order. As part of the order process, they each return a price for which they would supply the items in the order. A condition in the process determines which supplier is assigned the order.

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

3.2 Setting Up the Fusion Order Demo Application

This section describes how to prepare the environment to run the WebLogic Fusion Order Demo application.

3.2.1 Task 1: Install Oracle JDeveloper Studio

Install Oracle JDeveloper 11g Studio Edition to create the WebLogic Fusion Order Demo application. You can download Oracle JDeveloper from:

<http://www.oracle.com/technology/products/jdev/11/index.html>

Ensure that you download and install 11g and that it is the Studio Edition, not the Java Edition. You can verify these details in Oracle JDeveloper from the **Help > About** menu option.

In order to create and deploy SOA composite applications and projects, you must install the Oracle SOA Suite extension. For instructions on installing this extension for Oracle JDeveloper, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

3.2.2 Task 2: Install the Fusion Order Demo Application

Throughout this tutorial, you must view or use content from Fusion Order Demo in your Oracle JDeveloper environment. The Fusion Order Demo is contained within a ZIP file.

To access the ZIP file:

1. Download the Fusion Order Demo application ZIP file (FusionOrderDemo_R1PS4.zip). You can download the ZIP file from:

<http://www.oracle.com/technology/products/jdev/samples/fod/index.html>

2. Unzip the file to a temporary directory.

This tutorial refers to this directory as `DEMO_DOWNLOAD_HOME`.

3.2.3 Task 3: Install Oracle SOA Suite

To successfully deploy and run the Fusion Order Demo applications, you must complete an installation for Oracle SOA Suite. Specifically, the domain contains an Administration Server and a Managed Server.

Installing Oracle SOA Suite requires the following

- Creating schemas for Oracle SOA Suite in an Oracle database
- Installing Oracle WebLogic Server
- Configuring a domain in Oracle WebLogic Server to support Oracle SOA Suite, Oracle Enterprise Manager, and optionally, Oracle BAM. Oracle BAM is not required for the Fusion Order Demo, but if an Oracle BAM Server is configured, Oracle BAM adapters send data to the Oracle BAM Server.

After the domain is created, it contains an Administration Server to host Oracle Enterprise Manager Fusion Middleware Control for performing administrative tasks, a Managed Server to host deployed applications, and, if you configured Oracle BAM, a second Managed Server for the Oracle BAM Server.

For instructions on installing and configuring Oracle SOA Suite, see the *Oracle Fusion Middleware Installation Guide for Oracle SOA Suite*.

After successfully completing the installation process, perform the following additional configuration steps:

1. Enable the credentials that are included in the StoreFront module by adding a setting to the configuration file for the domain:

- a. Locate the configuration file set for the Oracle SOA Suite domain in the following directory:

```
(UNIX) MW_HOME/user_projects/domains/domain_name/bin/setDomainEnv.sh
(Windows) MW_HOME\user_projects\domains\domain_name\bin\setDomainEnv.cmd
```

- b. Add the following option to the JAVA_PROPERTIES (UNIX) or the SET JAVA_PROPERTIES (Windows) line:

```
-Djps.app.credential.override.allowed=true
```

For more information about setting this property, see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

- c. If the Oracle WebLogic Server Administration Server is running, stop it:

On UNIX, as the root user, change directories to directory *MW_HOME/user_projects/domains/domain_name/bin* and enter the following command:

```
./stopWebLogic.sh
```

On Windows, from the Windows **Start** menu, select **All Programs > Oracle WebLogic > User Projects > domain_name > Stop Admin Server**.

- d. Start the Administration Server:

On UNIX, from directory *MW_HOME/user_projects/domains/domain_name/bin*, enter the following command:

```
./startWebLogic.sh
```

On Windows, from the Windows **Start** menu, select **All Programs > Oracle WebLogic > User Projects > domain_name > Start Admin Server**.

When prompted on UNIX, enter your Oracle WebLogic Server user name and password. The password is not visible as you type.

The Administration Server is started when the command window displays the following messages:

```
<Server state changed to RUNNING>
<Server started in RUNNING mode>
```

Leave the command window open, although you may minimize it. The Administration Server is now running and ready for use.

- e. When the Administration Server is in `RUNNING` mode, start the Managed Servers, if they are not running. In a command window, enter the following command all on one line:

On UNIX, from directory `MW_HOME/user_projects/domains/domain_name/bin`, enter the following command:

```
./startManagedWebLogic.sh managed_server_name admin_url username password
```

On Windows, from directory `MW_HOME\user_projects\domains\domain_name\bin`, enter the following command:

```
startManagedWebLogic.cmd managed_server_name admin_url username password
```

Substitute the following values in [Table 3–1](#).

Table 3–1 *startManagedWebLogic Values*

Value	Description
<i>managed_server</i>	The name of the Managed Server. For example: soa_server1 bam_server1
<i>admin_url</i>	The URL of the Managed Server. For example: http://soahost:8001 http://soahost:9001 The port of the Managed Server for hosting SOA applications is typically 8001. The port of the Managed Server for Oracle BAM is typically 9001.
<i>username</i>	The Oracle WebLogic Server administrator. For example: weblogic
<i>password</i>	The password of the Oracle WebLogic Server administrator. For example: welcome1

2. If you are deploying remotely from one computer that has Oracle JDeveloper to another computer that has the Oracle SOA Suite installation with Oracle WebLogic Server, modify the `JAVA_HOME` and `PATH` environment variables on the computer with the Oracle SOA Suite installation.

Oracle JDeveloper requires changes to these variables for running the scripts that deploy the composite services. You set the `JAVA_HOME` variable to include the path to the Oracle WebLogic Server JDK, and set the `PATH` variable to include the path to the Oracle WebLogic Server `bin` directory for `ant`.

On UNIX, use the `export` command. For example:

```
export JAVA_HOME=$MW_HOME/jdk160_11
export PATH=$PATH:$MW_HOME/modules/org.apache.ant_1.7.0/bin
```

On Windows, perform the following steps to modify the variables:

- a. Open the Control Panel from the Windows Start menu and double-click the **System** icon.
- b. In the System Properties dialog, select the **Advanced** tab and click **Environment Variables**.
- c. In the Environment Variables dialog, locate the JAVA_HOME system variable and ensure that it is set to the location of the Oracle WebLogic Server JDK.

If there is no JAVA_HOME variable defined, click **New** and in the New System Variable dialog, enter a variable name of JAVA_HOME and a variable value pointing to the Oracle WebLogic Server JDK, such as C:\weblogic\jdk160_11. Click **OK** to set the new system variable.

- d. Double-click the Path system variable and ensure that it includes the path to the Oracle WebLogic Server ant\bin directory. If it does not, add the path to the end of the variable value. For example:

```
;C:\weblogic\modules\org.apache.ant_1.7.0\bin
```

Click **OK** to set the new system variable.

- e. Click **OK** twice more to dismiss the Environment Variables and the System Properties dialogs.

3.3 Taking a Look at the WebLogic Fusion Order Demo Application

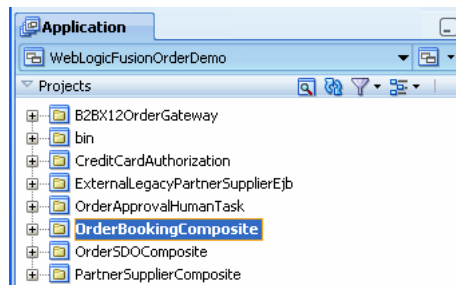
After you have set up the WebLogic Fusion Order Demo application, spend time viewing the WebLogic Fusion Order Demo artifacts in Oracle JDeveloper.

To open the WebLogic Fusion Order Demo in Oracle JDeveloper:

1. From the Oracle JDeveloper main menu, choose **File > Open**.
2. In the Open dialog, browse to `DEMO_DOWNLOAD_HOME/CompositeServices` and select **WebLogic Fusion Order Demo.jws**. Click **Open**.
3. When prompted to migrate files to the 11.1.1.5.0 format, click **Yes**. When the migration is complete, click **OK**.

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

Figure 3–2 Projects of WebLogic Fusion Order Demo Application



3.3.1 Project Applications of the WebLogic Fusion Order Demo Application

[Table 3–2](#) lists and describes the projects in the `WebLogicFusionOrderDemo` application workspace.

Table 3–2 Projects in the WebLogic Fusion Order Demo Application

Application	Description
<code>B2BX12OrderGateway</code>	This project contains a composite for Oracle B2B. This composite is not used in this guide.
<code>bin</code>	This project contains a build script for deploying all the SOA projects. It also contains templates for seeding JMS connector information, demo topics, and demo users.
<code>CreditCardAuthorization</code>	This project provides the service needed by OrderBookingComposite project to verify the credit card information of a customer.
<code>ExternalLegacyPartnerSupplierEjb</code>	This project provides an external system to provide price quotes.
<code>OrderApprovalHumanTask</code>	This project provides a task form for approving orders from the OrderBookingComposite project.
<code>OrderBookingComposite</code>	This project processes an order submitted in the Store Front module user interface. This project contains the main process for the WebLogic Fusion Order Demo application. It also uses the Oracle BAM adapter and Oracle BAM sensors to send active data into the Oracle BAM dashboard. This composite is not used in this guide.
<code>OrderSDOComposite</code>	This project simulates the StoreFrontService service of the Store Front module for testing purposes.
<code>PartnerSupplierComposite</code>	This project contains a composite containing both a BPEL process and spring context for obtaining a quote from a partner warehouse. It is referenced as a service from the composite for the OrderBookingComposite project. The quote request is routed to either the BPEL process or the spring component based on the amount.

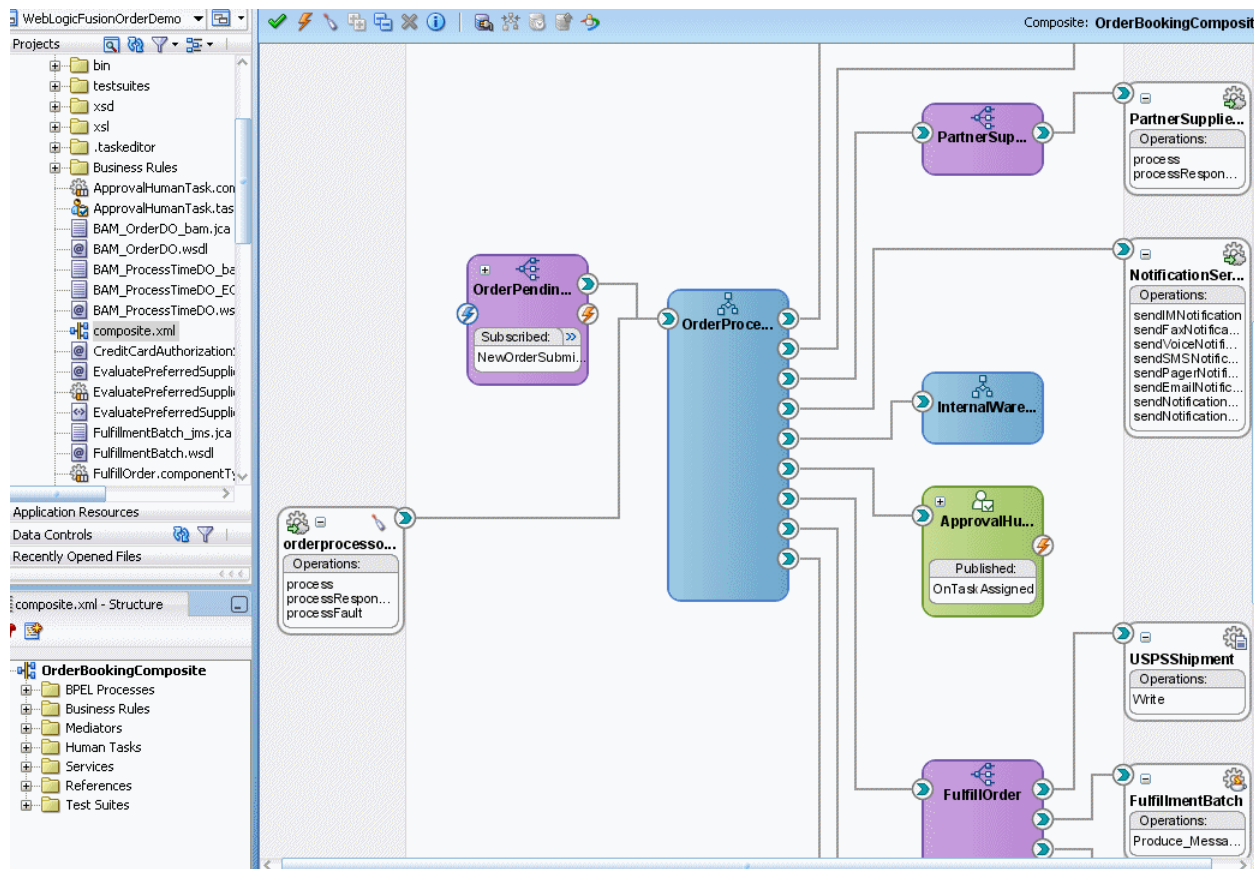
3.3.2 The composite.xml File

To understand how a composite is designed, examine the main project, `OrderBookingComposite`, in Oracle JDeveloper.

To view the composite.xml file:

1. In Application Navigator, expand **OrderBookingComposite > SOA Content**.
2. Select **composite.xml**.

The composite then appears in the SOA Composite Editor in Oracle JDeveloper, as shown in [Figure 3–3](#).

Figure 3–3 SOA Composite Editor

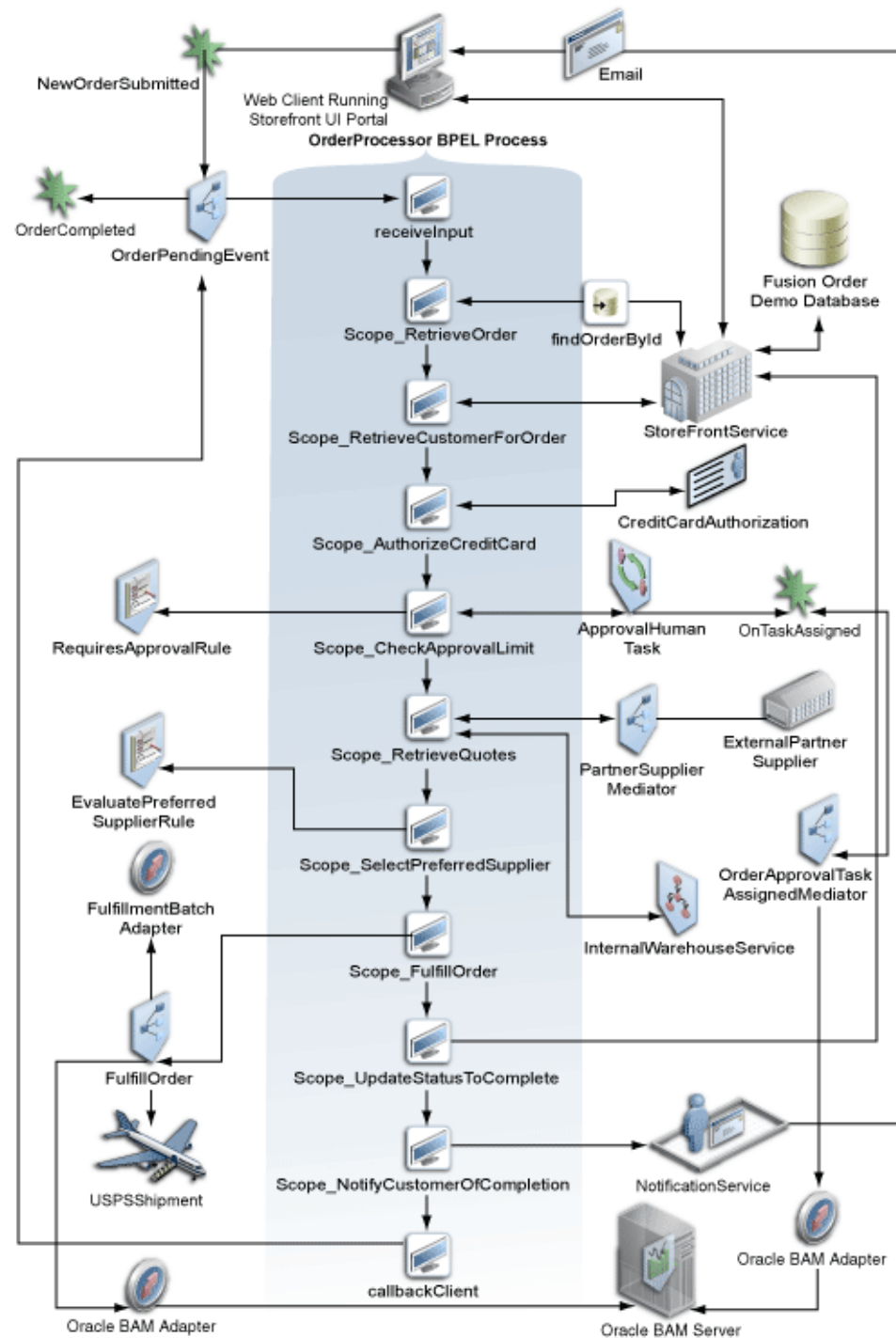
3.4 Understanding the OrderBookingComposite Flow

OrderBookingComposite is the main project of the WebLogic Fusion Order Demo application, containing a composite application for processing orders from Global Company. This composite demonstrates how services, both internal to an enterprise, and external at other sites, can be integrated using the SOA architecture paradigm to create one cohesive ordering system.

At the center of OrderBookingComposite composite is the OrderProcessor BPEL process. It orchestrates all the existing services in the enterprise for order fulfillment with the right warehouse, based on the business rules in the process.

Figure 3–4 shows an overview of the OrderBookingComposite composite for the WebLogic Fusion Order Demo application, followed by a step-by-step description of the composite flow for how the application processes an order.

Figure 3-4 OrderBookingComposite Flow



When a new customer registers in Global Company's storefront user interface, the web client sends the customer's information to the internal customer service application called StoreFrontService. StoreFrontService then stores the customer information in a database. The customer can then browse products, add them to their online shopping cart, and place the order. User ngreenbe is the only user not required to register before placing an order.

When a registered customer uses Global Company's storefront user interface, the user interface invokes the StoreFrontService and provides authentication. A registered user fills their shopping cart, and places an order. When the order is submitted, the following events take place:

After an order is placed, the following sequence occurs to complete the order:

1. Oracle ADF Business Component writes the order to a database with schema for Fusion Order Demo, and raises a NewOrderSubmitted event using the Event Delivery Network (EDN). The data associated with this event is the order ID.
2. Because the OrderPendingEvent Oracle Mediator subscribes to the NewOrderSubmitted event, the EDN layer notifies the OrderPendingEvent Oracle Mediator of the new order.
3. The OrderPendingEvent Oracle Mediator receives the order and routes the input order ID to the OrderProcessor BPEL process.
4. The OrderProcessor BPEL process receives the order ID from the database, using a bind entity activity to bind to the exposed Oracle ADF Business Component StoreFrontService service.

Some of the information about the order used later in the process is:

- Customer ID
 - Items the customer purchased
 - Credit card used
 - Shipping address chosen
5. The BPEL process initiates StoreFrontService, passing it the order ID, to retrieve information about the customer.
 6. The BPEL process then sends the purchase amount, credit card type, and credit card number to CreditCardAuthorizationService, which verifies if the customer's credit card is valid.

If the credit card is not valid, the BPEL process cancels the order.

If the credit card is valid, the BPEL process sends the order to the RequiresApprovalRule business rule to determine if the order requires approval by management.

7. The RequiresApprovalRule business rule evaluates if manual approval is required. The business rule contains a rule that requires manual approval for orders over \$2,000.
8. For those orders requiring manual approval, the BPEL process invokes the ApprovalHumanTask human task, which in turn performs the following:
 - Routes a message to an assignee named jstein, who then approves or disapproves the order.
 - Publishes the OnTaskAssigned event. The OrderApprovalTaskAssignedMediator Oracle Mediator subscribes to this event, and if an Oracle BAM Server is configured, it uses an Oracle BAM adapter to send the assignee ID jstein (based on the ECID) of the order to the Oracle BAM Server.
9. If the order is approved, the BPEL process sends the order information to the following suppliers in parallel to obtain a bid:

- Internal supplier by using the InternalWarehouseService BPEL process, also located in OrderBookingComposite
 - External supplier by using the PartnerSupplierMediator Oracle Mediator, which in turn routes to the ExternalPartnerSupplier BPEL process or SpringPartnerSupplierMediator spring component, located in another composite called PartnerSupplierComposite
10. The two suppliers respond with their bids, and the BPEL process send the bids to the EvaluatePreferredSupplierRule business rule.
 11. The EvaluatePreferredSupplierRule business rule chooses the supplier with the lower of the two bids.
 12. The BPEL process invokes the FulfillOrder Oracle Mediator, which performs the following four operations:
 - Stores the order in a temporary queue and uploads it to the fulfillment system in batch mode overnight
 - Routes the order to USPS
 - If an Oracle BAM Server is configured, it uses an Oracle BAM adapter to send data about the order (based on order ID) to the Oracle BAM Server.
 - If an Oracle BAM Server is configured, it uses an Oracle BAM adapter to send data about the time for the order to process (based on the instance ID) to the Oracle BAM Server.
 13. Once the order is fulfilled, the BPEL process sets the order to complete.
 14. The BPEL process invokes the NotificationService service, which sends the customer an email notification with the purchase order information.
 15. When the order completes, the OrderPendingEvent Oracle Mediator publishes the OrderCompleted business for the OrderProcessor process.

While not depicted in [Figure 3–4](#), the OrderBookingComposite composite provides the following processing flow for approved orders:

1. The UpdateOrderStatus Oracle Mediator performs the following:
 - Publishes business event OrderUpdateEvent and sends the order ID to the OrderProcessor BPEL process.
 - If an Oracle BAM Server is configured, it uses an Oracle BAM adapter to send data about the order ID and order status to the Oracle BAM Server.
2. The OrderUpdateEventMediator Oracle Mediator subscribes to business event OrderUpdateEvent, sends the order ID to StoreFrontService, and waits for the StoreFrontService to respond with updated details about the order.

To aid with the tracking of an order, the OrderBookingComposite composite contains sensors to provide a method for implementing trackable fields on messages. For example, the CreditCardAuthorization service has a composite sensor that indicates if the credit card was authorized. In addition, the OrderProcessor BPEL process also uses sensors for various activities. For example, the Scope_AuthorizeCreditCard scope in the OrderProcessor BPEL process, which verifies that the customer has acceptable credit using the CreditCardAuthorizationService service, uses a sensor for tracking. When you monitor instances of a composite through Oracle Enterprise Manager Fusion Middleware Control, you can monitor the sensors for both the composite and the BPEL process.

In the remaining sections of this chapter, deploy and run the Fusion Order Demo. As a part of it running it, use Oracle Enterprise Manager Fusion Middleware Control to monitor orders processed by the OrderBookingComposite composite. When you monitor an order, you can also view the composite sensors and activity sensors.

3.5 Deploying Fusion Order Demo

This section describes how to deploy the Fusion Order Demo applications in the partition.

3.5.1 Task 1: Create a Connection to an Oracle WebLogic Server

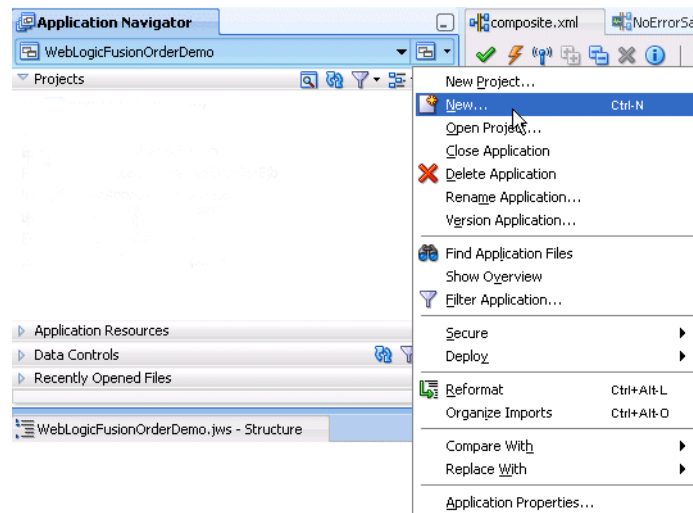
To create a connection to an Oracle WebLogic Server:

1. Start Oracle JDeveloper:

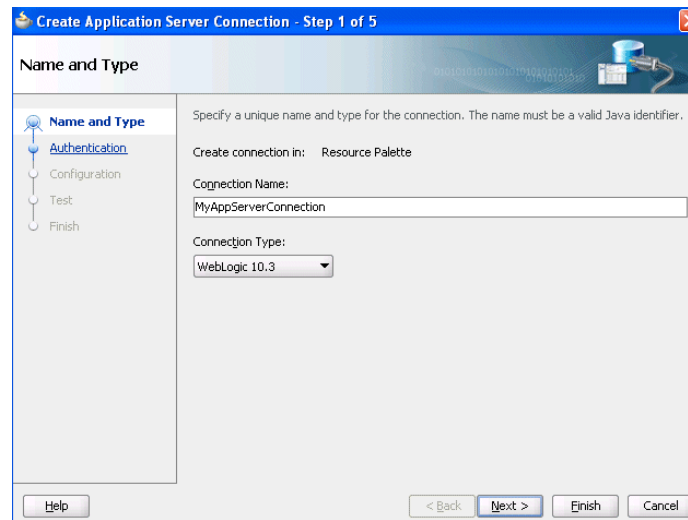
```
(UNIX) ORACLE_HOME/jdev/bin/jdev
(Windows) JDEV_ORACLE_HOME\jdeveloper\JDev\bin\jdev.exe
```

2. From the **Application Menu**, select **New**, as shown in [Figure 3–5](#).

Figure 3–5 Application Menu



3. In the New Gallery dialog, in the **Categories** tree, select **General**, and then **Connections**.
4. Select **Application Server Connection** and click **OK**.
The Create Application Server Connection Type page displays.
5. Enter a unique name for the connection in the **Connection Name** field and select **WebLogic 10.3** from the **Connection Type** list. [Figure 3–6](#) provides details.

Figure 3–6 Create Application Server Connection**6. Click Next.**

The Authentication page is displayed.

7. Enter weblogic for the User Name and the password for that administrator in the Password field.**8. In the Configuration page, enter the details shown in [Table 3–3](#).****Table 3–3 Configuration Page Fields and Values**

Application	Description
Weblogic Hostname (Administration Server)	Name of the DNS name or IP address of the Administration Server of the Oracle WebLogic Server
Port	The address of the port on which the Administration Server is listening for requests (7001 by default)
Weblogic Domain	The domain name for Oracle WebLogic Server

9. Click Next.

The Test page displays.

10. Click Test Connection.

The following message should appear:

```

Testing JSR-88                ... success.
Testing JSR-88-LOCAL          ... success.
Testing JNDI                   ... success.
Testing JSR-160 DomainRuntime ... success.
Testing JSR-160 Runtime        ... success.
Testing JSR-160 Edit           ... success.
Testing HTTP                   ... success.
Testing Server MBeans Model    ... success.

```

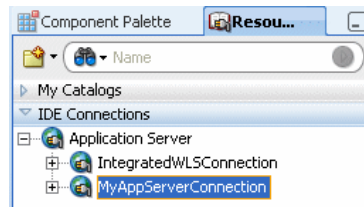
8 of 8 tests successful.

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

11. Click Finish.

12. In the Resource Palette, under **IDE Connections**, expand **Application Server** to see the application server connection that you created. [Figure 3–7](#) provides details.

Figure 3–7 Resource Palette



3.5.2 (Optional) Task 2: Create a Connection to the Oracle BAM Server

If you configured an Oracle BAM Server during installation, create a connection to it.

To create a connection to an Oracle BAM Server:

1. From the **Application Menu**, select **New**.
2. In the New Gallery dialog, in the **Categories** tree, select **General**, and then **Connections**.
3. Select **BAM Connection** and click **OK**.
The BAM Connection Wizard displays.
4. Ensure that **Application Resources** is selected.
5. Provide a name for the connection.
6. Click **Next**.
7. Enter `weblogic` for the **User Name** and the password for that administrator in the **Password** field.
8. Enter the connection information about the Oracle BAM Server host described in [Table 3–4](#).

Table 3–4 Oracle BAM Server Connection Information

Field	Description
BAM Web Host	Enter the name of the host on which the Oracle BAM Report Server and web applications are installed. In most cases, the Oracle BAM web applications host, Oracle BAM Server host, and the Oracle WebLogic Server 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. For example: <code>weblogic</code>
Password	Enter the password of the user name.
HTTP Port	Enter the port number or accept the default value of 9001. This is the HTTP port for the Oracle BAM web applications host.
JNDI Port	Enter the port number or accept the default value of 9001. The JNDI port is for the Oracle BAM report cache, which is part of the Oracle BAM Server.

Table 3–4 (Cont.) Oracle BAM Server Connection Information

Field	Description
Use HTTPS	Select this checkbox to use secure HTTP (HTTPS) to connect to the Oracle BAM Server during design time. Otherwise, HTTP is used.

9. Click Next.

The Test page displays.

10. Click Test Connection.

The following message should appear:

```
Testing HTTP connection ... success.
Testing Data Object browsing ... success.
Testing JNDI connection ... success.
```

3 of 3 tests successful.

11. Click Finish.

3.5.3 Task 3: Install the Schema for the Fusion Order Demo Application

To install the schema for the sample application:

1. Start Oracle JDeveloper 11g, and from the main menu choose **File > Open**.
2. In the Open dialog, browse to `DEMO_DOWNLOAD_HOME/Infrastructure` and select **Infrastructure.jws**.
3. Click **Open**.
4. When prompted to migrate files to the 11.1.1.5.0 format, click **Yes**. When the migration is complete, click **OK**.
5. In the Application Navigator, expand **MasterBuildScript** and then **Resources**, and double-click **build.properties**.
6. In the editor, modify the following properties shown in [Table 3–5](#) for your environment.

Table 3–5 Properties Required to Install the Fusion Order Demo Application

Field	Description
<code>jdeveloper.home</code>	The root directory where you have Oracle JDeveloper 11g installed. For example: <code>C:/JDeveloper/11</code>
<code>jdbc.urlBase</code>	The base JDBC URL for your database in the format <code>jdbc:oracle:thin:@<yourhostname></code> . For example: <code>jdbc:oracle:thin:@localhost</code>
<code>jdbc.port</code>	The port for your database. For example: <code>1521</code>
<code>jdbc.sid</code>	The SID of your database. For example: <code>ORCL</code> or <code>XE</code>
<code>db.adminUser</code>	The administrative user for your database. For example: <code>system</code>

Table 3–5 (Cont.) Properties Required to Install the Fusion Order Demo Application

Field	Description
db.demoUser.tablespace	The tablespace name for the Fusion Order Demo users. For example: USERS

- From the JDeveloper main menu, choose **File > Save All**.
- In the Application Navigator, under the **Resources** node, right-click **build.xml** and choose **Run Ant Target > buildAll**.
- When prompted, enter the administrative-user password for your database.

The **buildAll** command then creates the FOD user and populates the tables in the FOD schema. In the Apache Ant - Log, a series of SQL scripts display, followed by:

```
buildAll:
BUILD SUCCESSFUL
Total time: nn minutes nn seconds
```

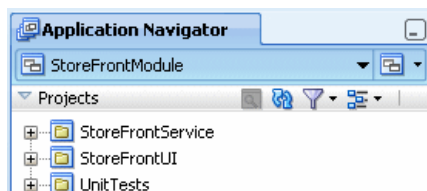
For more information on the demo schema and scripts, see the `README.txt` file in the **MasterBuildScript** project.

3.5.4 Task 4: Set the Configuration Property for the Store Front Module

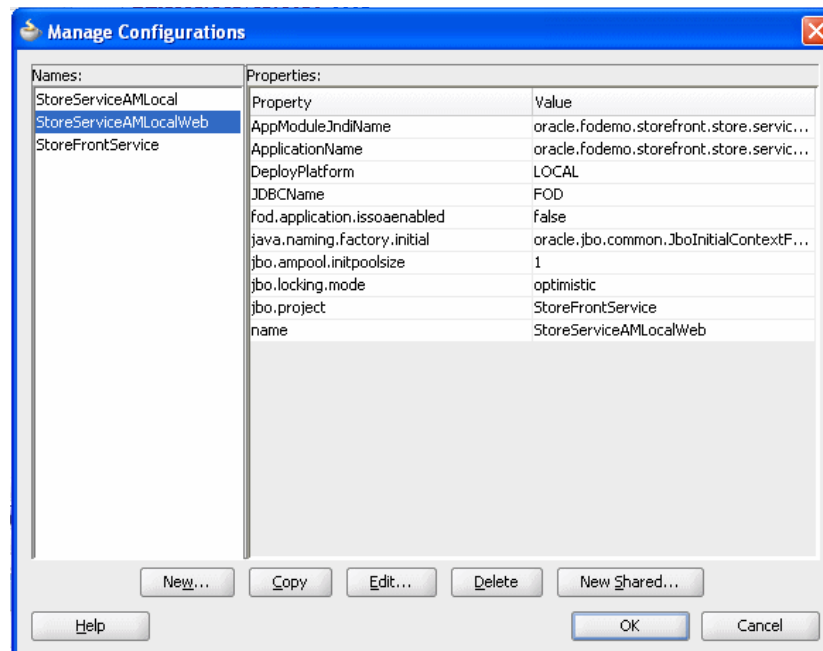
You can deploy the Store Front module as a simple web application or as part of a SOA environment. There is a property defined in the service portion of the Store Front module that is used within one of its pages to determine whether the Submit Order button fires an event that launches a BPEL process. When using the Store Front module within a SOA environment, you must change the default value for this property.

- Choose **File > Open**.
- In the Open dialog, browse to `DEMO_DOWNLOAD_HOME/StoreFrontModule` and select **StoreFrontModule.jws**. Click **Open**.
- When prompted to migrate files to the 11.1.1.5.0 format, click **Yes**. When the migration is complete, click **OK**.

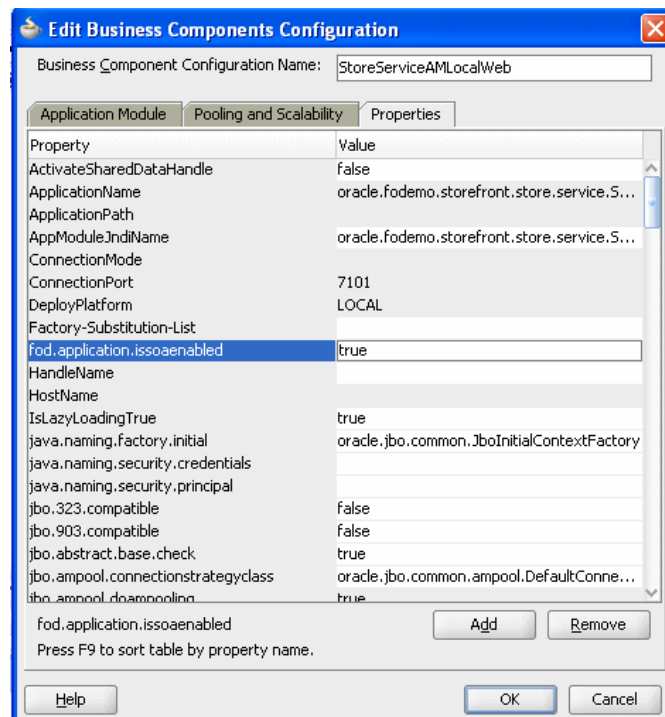
Figure 3–8 shows the Application Navigator after you open the file for the application workspace.

Figure 3–8 Application Navigator with StoreFrontModule

- In the Application Navigator, expand **StoreFrontService > Application Sources > oracle.fodemo.storefront > store > service**.
- Right-click **StoreServiceAM** and select **Configurations**.
- In the Manage Configurations dialog, select **StoreServiceAMLocalWeb** in the **Names** list, and then click **Edit**. Figure 3–9 provides details.

Figure 3–9 StoreServiceAMLocalWeb

7. In the Edit Business Components Configuration dialog, select the **Properties** tab and the **fod.application.issoaenabled** property. This property specifies whether the application is being deployed to a SOA environment.
8. Change the value of the **fod.application.issoaenabled** property to **true**, and then click **OK**. Figure 3–10 provides details.

Figure 3–10 fod.application.issoaenabled

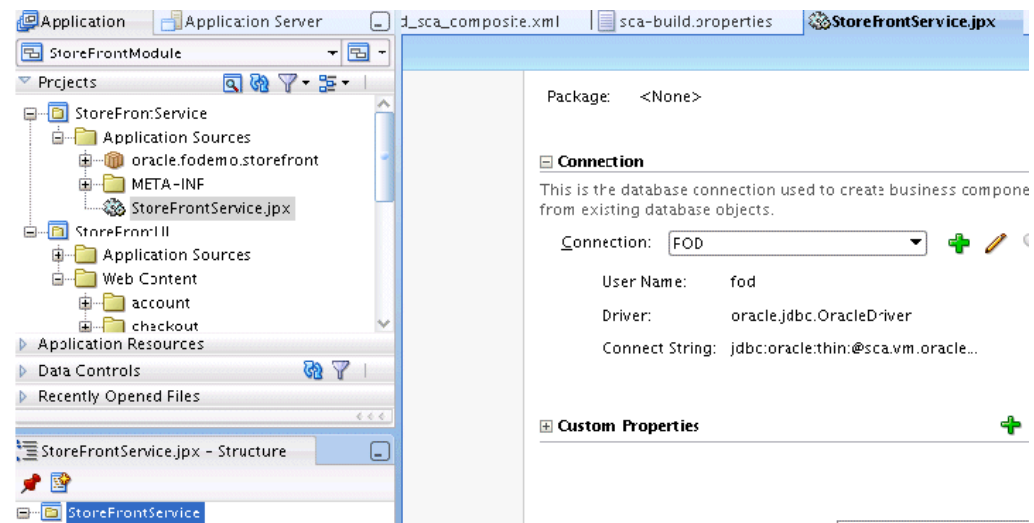
9. Click **OK**.
10. In the Manage Configurations dialog, click **OK**.

3.5.5 Task 5: Edit the Database Connection

Edit the database connection details to point to the correct host name and database SID.

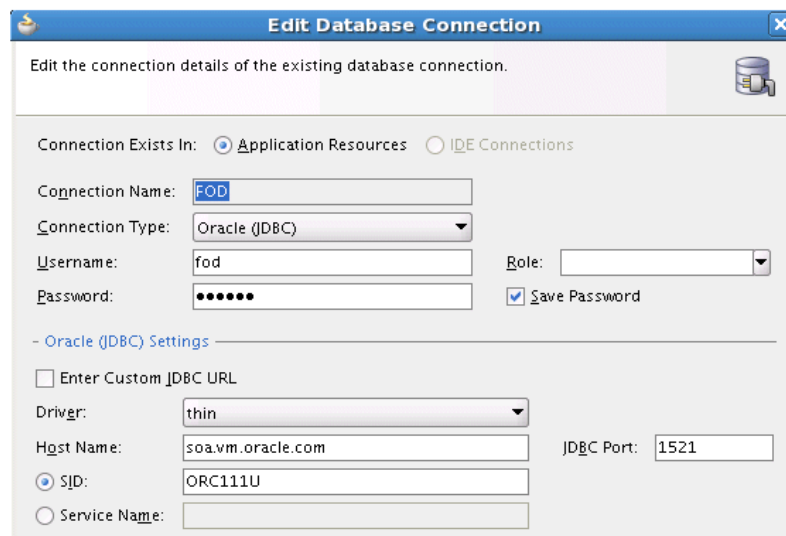
1. In the Application Navigator, expand **StoreFrontService > Application Sources**.
2. Double-click **StoreFrontService.jpx**.
3. To the right of the **Connection** field, click the **Edit** icon, as shown in [Figure 3–11](#).

Figure 3–11 Connection



4. Edit the connect string for the FOD database connection by replacing the values in the **Host Name** and **SID** fields with the correct host and SID. [Figure 3–12](#) provides details.

Figure 3–12 Host Name and SID Fields Modifications



3.5.6 Task 6: Deploy the Store Front Module

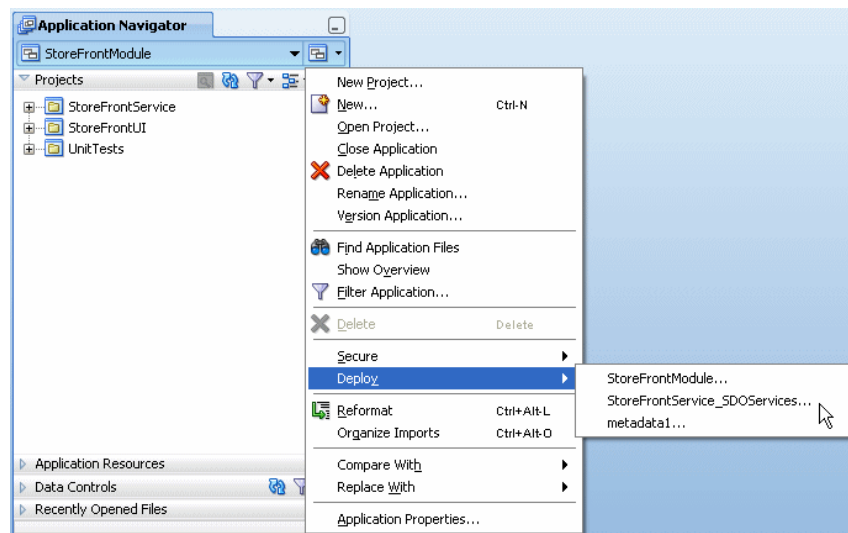
To deploy the Store Front module, you first deploy services and then to deploy the application itself.

During deployment, Oracle JDeveloper creates the .jar and .war files and then assembles the .ear file, as specified in the deployment profiles. After the file is assembled, Oracle JDeveloper deploys the .ear file and unpacks it in a directory on the application server. The directory that is used is dependent on the target environment.

To deploy the Store Front module:

1. Deploy the services used by the Store Front module to send orders to the **OrderBookingComposite** composite.
 - a. From the Application menu, choose **Deploy > StoreFrontModule_SDOservices**. [Figure 3–13](#) provides details.

Figure 3–13 *StoreFrontService_SDOservices*



- b. In the Deployment Action page of the Deploy StoreFrontService_SDOservices dialog, select **Deploy to Application Server**, and then click **Next**.
 - c. In the Select Server page, select **MyAppServerConnection**. You created this connection in [Section 3.5.1, "Task 1: Create a Connection to an Oracle WebLogic Server."](#)
 - d. Deselect option **Deploy to all server instances in the domain**, and then click **Next**.
 - e. In the Server Instances page, select the Managed Server for the Oracle WebLogic Server, such as **soa_server**, and click **OK**.
 - f. In the Summary page, click **Finish**.
 - g. View the messages that display in the Deployment log window at the bottom of Oracle JDeveloper to ensure deployment was successful.
 2. Deploy the Store Front module. From the Application menu, select **Deploy > StoreFrontModule > to > MyAppServerConnection**.
 - a. From the Application menu, choose **Deploy > StoreFrontModule**.

- b. In the Deployment Action page of the Deploy StoreFrontModule dialog, select **Deploy to Application Server**, and then click **Next**.
- c. In the Select Server page, select **MyAppServerConnection**.
- d. Deselect option **Deploy to all server instances in the domain**, and then click **Next**.
- e. In the Server Instances page, select the Managed Server for the Oracle WebLogic Server, such as **soa_server**, and click **Next**.
- f. In the Summary page, click **Finish**.
- g. View the messages that display in the Deployment log window at the bottom of Oracle JDeveloper to ensure that deployment was successful.

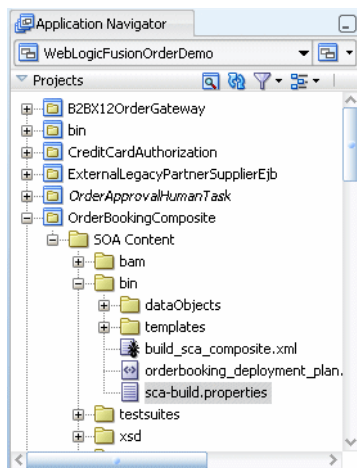
3.5.7 Task 7: Deploy the WebLogic Fusion Order Demo Application

In this task, you deploy the WebLogic Fusion Order Demo application to an Oracle SOA Suite installation, containing an Oracle WebLogic Server domain with an Administration Server and a Managed Server.

To deploy the WebLogic Fusion Order Demo application:

1. In the Application Navigator, select **WebLogicFusionOrderDemo**.
2. If you configured an Oracle BAM server during installation, perform the following steps:
 - a. From the Application Navigator, expand **OrderBookingComposite**, then **SOA Content**, and then **bin**. Double-click **sca-build.properties**. [Figure 3-14](#) provides details.

Figure 3-14 Navigating to *sca-build.properties*



- b. In the editor, modify the following properties shown in [Table 3-6](#) for the Oracle BAM environment.

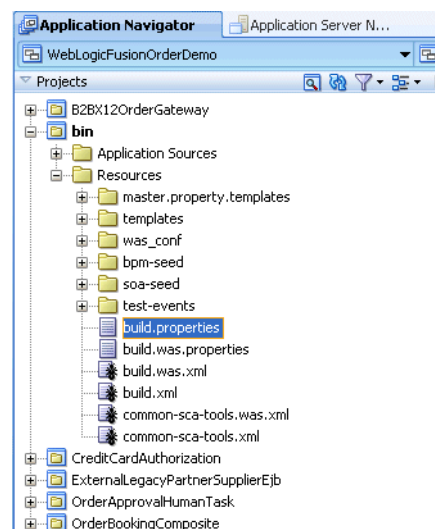
Table 3-6 Properties Required for Oracle BAM

Field	Description
<code>enable.bam.sensors</code>	<code>true</code>
	Set to <code>true</code> to enable sensors for Oracle BAM.

Table 3–6 (Cont.) Properties Required for Oracle BAM

Field	Description
<code>seed.bam.do</code>	<code>true</code> Set to <code>true</code> to seed data objects, alerts, and reports for Oracle BAM. After deployment is done, set this value back to <code>false</code>. If this parameter is set to <code>true</code> after initial deployment and you redeploy at a later time, then the data objects, alerts, and reports reseed. Therefore, after initial deployment, set this parameter to <code>false</code>.
<code>bam.server.host</code>	The DNS name or IP address of the Managed Server for Oracle BAM. For example: <code>soahost</code>
<code>bam.server.port</code>	The port of the Managed Server for Oracle BAM. For example: <code>9001</code>
<code>bam.server.username</code>	The Oracle WebLogic Server administrator. For example: <code>weblogic</code>
<code>bam.server.password</code>	The password of the Oracle WebLogic Server administrator. For example: <code>welcome1</code>

- c. From the Oracle JDeveloper main menu, choose **File > Save All**. Keep the **sca-build.properties** tab open, so you can modify the `seed.bam.do` parameter to `false` after deployment.
3. In the editor, perform the following steps for the **WebLogicFusionOrderDemo** application:
 - a. From the Application Navigator, expand **bin**, and then **Resources**. Double-click **build.properties**. [Figure 3–15](#) provides details.

Figure 3–15 Navigating to build.properties

- b. In the editor, modify the following properties shown in [Table 3–7](#) for the **WebLogicFusionOrderDemo** application.

Table 3–7 Properties Required for the WebLogic Fusion Order Demo Application

Field	Description
oracle.home	The root directory in which you have Oracle JDeveloper 11g installed. For example: C:\Oracle\Middleware\jdeveloper\
soa.only.deployment	false You set this property to true if you are using the OrderSDOComposite composite to place orders. This guide assumes you are using the Store Front Module to place orders. Therefore, you must modify this property to false.
admin.server.host	The DNS name or IP address of the Administration Server for Oracle SOA Suite for hosting applications. For example: soahost
admin.server.port	The port of the Administration Server. For example: 8001
managed.server	The DNS name or IP address of the Managed Server for Oracle SOA Suite for hosting applications. For example: soahost
managed.server.port	The port of the Managed Server for Oracle SOA Suite for hosting applications. For example: 8001
server.user	The Oracle WebLogic Server administrator. For example: weblogic
server.password	The password of the Oracle WebLogic Server administrator. For example: welcome1
server.targets	The name of the Managed Server. For example: soa_server
soa.server.oracle.home	The location of where to store the deployment plans for the adapters. For example: C:\AS11gR1SOA
foreign.mds.type	The location of the Oracle Metadata Repository. Leave the value to db and supply values for the mds.db.userid, mds.db.password, and mds.db.url parameters to specify the location of the MDS Repository. Set the value to leave the default value to jdev. You do not have to specify the values for the following parameters:
soa.partition.name	The partition in which to deploy the composites. For example: soaFusionOrderDemo

4. From the JDeveloper main menu, choose **File > Save All**.
5. In the Application Navigator, under the **Resources** node, right-click **build.xml** and choose **Run Ant Target** and select the following ant targets in the specified sequential order shown in [Table 3–8](#).

Table 3–8 ant Targets to Deploy the WebLogic Fusion Order Demo Application

Target	Description
1. validateFodConfigSettings	This script validates the server settings, checks if the servers are up, and also validates the MDS settings. If this script returns without error, proceed with target <code>server-setup-seed-deploy-test</code> .
2. server-setup-seed-deploy-test	This script calls the following targets: ■ <code>compile-deploy-all</code> compiles, builds, and deploys all the SOA composites to the Managed Server. ■ <code>seedFodJmsResources</code> populates the JMS resources for the Fulfillment mediator. ■ <code>seedDemoUsers</code> adds <code>jstein</code> as the user to approve orders for over \$2,000. When you run the demo, you place an order for \$2,000 and log in to Oracle BPM Worklist as <code>jstein</code> and approve the order.

In the **Apache Ant - Log**, you should see the following message when the target successfully completes:

```
BUILD SUCCESSFUL
Total time: nn minutes nn seconds
```

If you set up Oracle BAM after you run target `server-setup-seed-deploy-test`, you can still configure Oracle BAM for Fusion Order Demo by running one of these targets:

- Rerun target `server-setup-seed-deploy-test`.
 - From the Application Navigator, right-click **build_sca_composite.xml**, (**OrderBookingComposite** > **SOA Content**) choose **Run Ant Target**, and then select **seedBAMServerObjects**.
6. Go back to the **sca-build.properties** tab and modify the `seed.bam.do` parameter to `false`.
 7. From the JDeveloper main menu, choose **File > Save All**.

3.6 Running Fusion Order Demo

You begin the ordering process in the storefront user interface, where you submit an orders.

When an order is submitted, the Application Development Framework Business Component writes the order to the database and raises an `NewOrderSubmitted` business event using the Events Delivery Network (EDN). The `OrderPendingEvent` mediator subscribes this event, and initiates the main BPEL process, `OrderProcessor`, to process the order.

After you submit an order, you use Oracle Enterprise Manager Fusion Middleware Control for the Oracle SOA Suite installation to monitor how the `OrderProcessor` BPEL process orchestrated the orders. If you submit an order for more than \$2,000, you can monitor how it requires human approval.

The instructions for placing orders and monitoring them in detail with Fusion Middleware Control are available from Oracle Technology Network:

http://download.oracle.com/otn_hosted_doc/jdeveloper/doc/11/runningfod_notes.pdf

3.7 Viewing Data Sent to Oracle BAM Server

If you configured an Oracle BAM server and a Managed Server for it, you can use the Oracle BAM Architect to view data sent to the server. For more information about using Oracle BAM applications, including Oracle BAM Architect, see *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring*.

3.8 Undeploying the Composites for the WebLogic Fusion Order Demo Application

To undeploy the WebLogic Fusion Order Demo composite applications:

1. Access the Undeploy SOA Composite wizard in Fusion Middleware Control through the options described in [Table 3–9](#).

Table 3–9 Options to Access Undeploy SOA Composite Wizard

From the SOA Infrastructure Menu...	From the SOA Folder in the Navigator...	From the SOA Infrastructure Home Page...	From the SOA Composite Menu...
1. Select SOA Deployment > Undeploy. The Select Composite page appears. 2. In the SOA Composite Deployments section, select OrderBookingComposite and PartnerSupplierComposite to undeploy them, and click Next.	1. Right-click soa-infra. 2. Select SOA Deployment > Undeploy. The Select Composite page appears. 3. In the SOA Composite Deployments section, select OrderBookingComposite and PartnerSupplierComposite to undeploy, and click Next.	1. Click the Deployed Composites tab. 2. In the Composite table, select both OrderBookingComposite and PartnerSupplierComposite. 3. Above the Composite table, click Undeploy.	1. Select SOA Deployment > Undeploy.

The Confirmation page appears.

2. Click **Undeploy**. Note that you are warned if you are about to undeploy the last remaining revision of a deployed composite application.

Processing messages display.

3. When undeployment has completed, click **Close**.

Part II

Using the BPEL Process Service Component

This part describes the BPEL process service component.

This part contains the following chapters:

- [Chapter 4, "Getting Started with Oracle BPEL Process Manager"](#)
- [Chapter 5, "Introduction to Interaction Patterns in a BPEL Process"](#)
- [Chapter 6, "Manipulating XML Data in a BPEL Process"](#)
- [Chapter 7, "Invoking a Synchronous Web Service from a BPEL Process"](#)
- [Chapter 8, "Invoking an Asynchronous Web Service from a BPEL Process"](#)
- [Chapter 9, "Using Parallel Flow in a BPEL Process"](#)
- [Chapter 10, "Using Conditional Branching in a BPEL Process"](#)
- [Chapter 11, "Using Fault Handling in a BPEL Process"](#)
- [Chapter 12, "Transaction and Fault Propagation Semantics in BPEL Processes"](#)
- [Chapter 13, "Incorporating Java and Java EE Code in a BPEL Process"](#)
- [Chapter 14, "Using Events and Timeouts in BPEL Processes"](#)
- [Chapter 15, "Coordinating Master and Detail Processes"](#)
- [Chapter 16, "Using the Notification Service"](#)
- [Chapter 17, "Using Oracle BPEL Process Manager Sensors"](#)

Getting Started with Oracle BPEL Process Manager

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

This chapter includes the following sections:

- [Section 4.1, "Introduction to the BPEL Process Service Component"](#)
- [Section 4.2, "Introduction to Activities"](#)
- [Section 4.3, "Introduction to Partner Links"](#)
- [Section 4.4, "Creating a Partner Link"](#)
- [Section 4.5, "Introduction to Technology Adapters"](#)
- [Section 4.6, "Introduction to BPEL Process Monitors"](#)

4.1 Introduction to the BPEL Process Service Component

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

4.1.1 How to Add a BPEL Process Service Component

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

To add a BPEL process service component:

1. Follow the instructions in [Table 4-1](#) to start Oracle JDeveloper.

Table 4-1 Starting Oracle JDeveloper

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

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

As a service component in an existing SOA composite application:

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

In a new application:

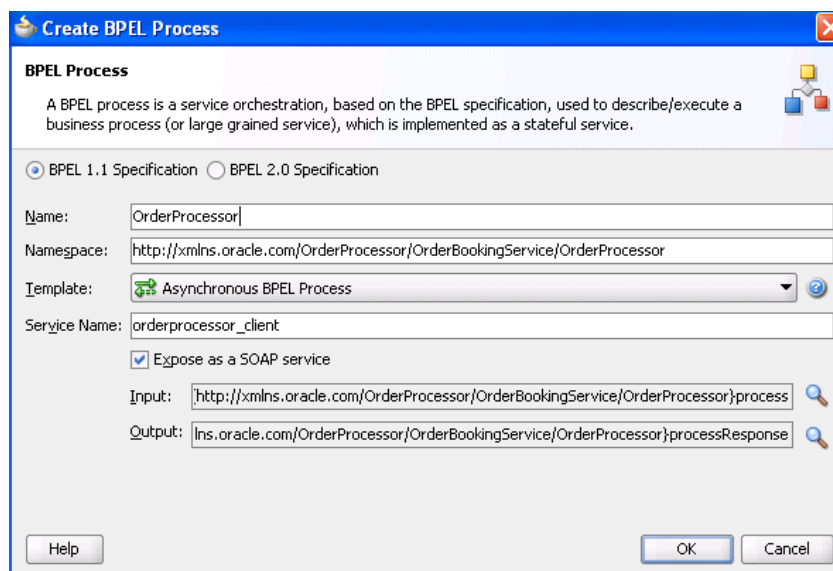
- a. From the Application Navigator, select **File > New > Applications > SOA Application**.
This starts the Create SOA Application wizard.
- b. In the Application Name dialog, enter an application name in the **Application Name** field.
- c. In the **Directory** field, enter a directory path in which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

Each method causes the Create BPEL Process dialog shown in [Figure 4–1](#) to appear.

3. Provide the required details (including BPEL process name and whether you want to create a BPEL project that supports the BPEL 1.1 or BPEL 2.0 specification). Click **Help** for details about the types of BPEL processes you can create.

Note: You cannot use BPEL 1.1 and BPEL 2.0 syntax in the same .bpel file. However, you can include BPEL 1.1 and BPEL 2.0 projects in the same SOA composite application.

Figure 4–1 Create BPEL Process Dialog



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

- A process name that begins with a number (for example, 1SayHello)
- A process name that includes a dash (for example, Say-Hello)

- Two processes with the same name, but with different capitalization (for example, SayHello and sayhello).

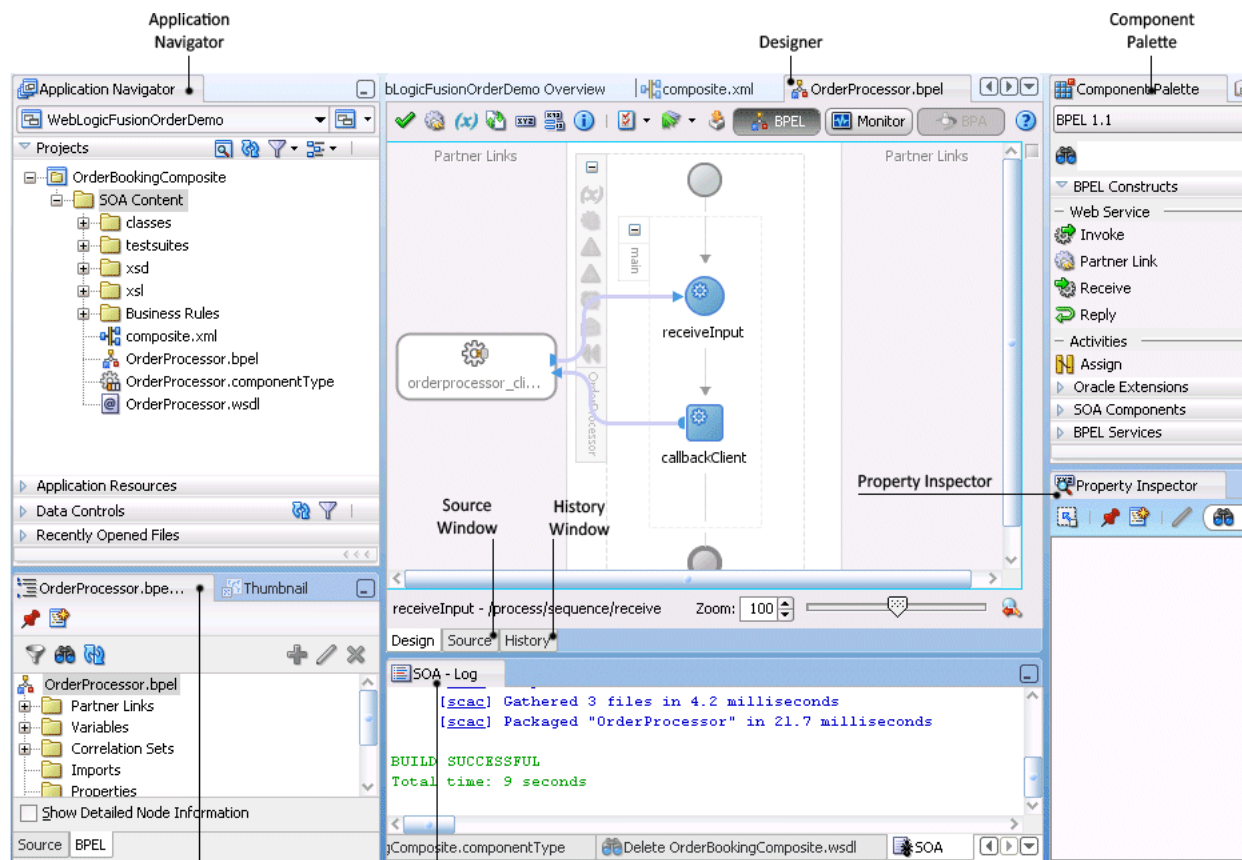
This is particularly important for business intelligence (BI) data object names, which are generated on the Oracle BAM server in all upper case format. For example, if you create a BPEL process named BPELProcess1, a BI name of BI_DEFAULT_PROJECT1_BPELPROCESS1 is generated for the Oracle BAM BI data object after deployment. If you create two BPEL processes, BPELProcess1 and BPELPROCESS1, the same BI data object name is generated.

- A process name that exceeds 500 characters.
- A non-ASCII process name. The BPEL process name is used in directory and file names of the SOA project, which can cause problems.

4. Click OK.

Oracle BPEL Designer displays the sections shown in [Figure 4-2](#).

Figure 4-2 Oracle BPEL Designer Sections



Each section of this view enables you to perform specific design and deployment tasks. [Table 4-2](#) identifies the sections listed in [Figure 4-2](#).

Table 4–2 Oracle JDeveloper Sections

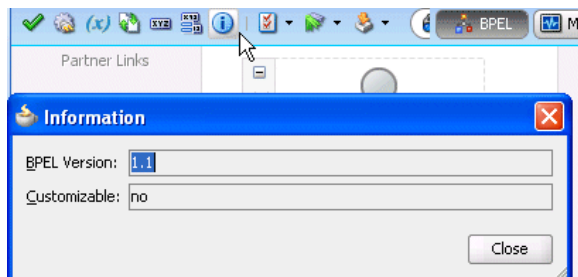
Element	Description
Application Navigator	Displays the process files of a SOA project. Key files include the following: ■ composite.xml Describes the entire SOA composite application. For more information about this file, see Section 2.1.2, "What Happens When You Create a SOA Application and Project." ■ .bpel Depending upon the process type you selected, initially contains a minimal set of activities (if you selected to create an asynchronous process, then receive and invoke activities appear). You add syntax to this file when you drag activities, create variables, create partner links, and so on. ■ .componentType Describes the services and references for the BPEL process service component. ■ .wsdl The Web Services Description Language (WSDL) client interface, which defines the input and output messages for this BPEL process flow, the supported client interface and operations, and other features. This functionality enables the BPEL process flow to be called as a service. ■ monitor.config Defines runtime and deployment properties needed to connect with Oracle BAM Server to create the Oracle BAM data objects and dashboards.
Designer	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 Editor. As you design the BPEL process service component by dragging activities, creating partner links, and so on, the Design window changes.
Component Palette	Displays the available activities to add to the BPEL process service component. Activities are the building blocks. The BPEL Constructs and Oracle Extensions selections of the Component Palette display a set of activities that you drag into the designer of the BPEL process service component. The Component Palette displays only those pages relevant to the state of the designer. BPEL Constructs or Oracle Extensions 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.

Table 4–2 (Cont.) Oracle JDeveloper Sections

Element	Description
Structure window	Provides a structural view of the data in the BPEL process service component currently selected in the designer. 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, and partner links. ■ Editing activities in the BPEL process flow sequence that displays in the designer
Log window	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.
Source window	View the syntax inside the BPEL process service component files. As you drag activities and partner links, and perform other tasks, the syntax in these source files is immediately updated to reflect these changes.
History window	Displays the revision history of a file and read-only and editable versions of a file side-by-side.
Property Inspector	Displays details about an activity. Single-click an activity in the Design window to display details.

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

5. Click the icon above the Oracle BPEL Designer to view the BPEL project version (either 1.1 or 2.0). [Figure 4–3](#) provides details.

Figure 4–3 BPEL Project Version

4.2 Introduction to Activities

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

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

Figure 4-4 Assign Activity



- An invoke activity enables you to invoke a service (identified by its partner link) and specify an operation for this service to perform. [Figure 4-5](#) shows an invoke activity.

Figure 4-5 Invoke Activity



- A receive activity waits for an asynchronous callback response message from a service. [Figure 4-6](#) shows a receive activity. A receive activity is also used when a process is started asynchronously through a partner link.

Figure 4-6 Receive Activity



[Figure 4-7](#) shows an example of a property window (for this example, an invoke activity). In this example, you invoke a partner link named **StoreFrontService** and define its attributes.

Figure 4–7 Invoke Activity Example

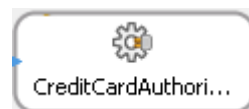
The invoke activity enables you to specify an operation you want to invoke for the service (identified by its partner link). The operation can be one-way or request-response on a port provided by the service. You can also automatically create variables in an invoke activity. An invoke activity invokes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

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

For more information about activities, see [Appendix A, "BPEL Process Activities and Services."](#)

4.3 Introduction to Partner Links

A partner link enables you to define the external services with which the BPEL process service component is to interact. You can define partner links as services or references (for example, through a JCA adapter) in the SOA Composite Editor or within a BPEL process service component in Oracle BPEL Designer. [Figure 4–8](#) shows the partner link icon (in this example, named **CreditCardAuthorizationService**).

Figure 4–8 PartnerLink Icon

A partner link type characterizes the conversational relationship between two services by defining the roles played by each service in the conversation and specifying the port type provided by each service to receive messages within the conversation.

[Figure 4–9](#) shows an example of the attributes of a partner link for a service.

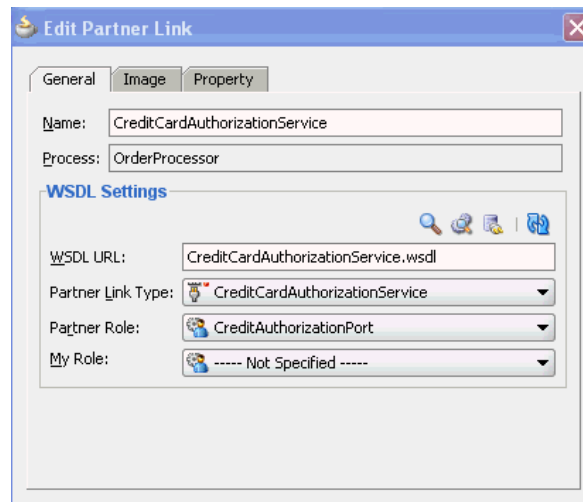
Figure 4–9 Partner Link Dialog

Table 4–3 describes the fields of this dialog.

Table 4–3 Create Partner Link Dialog Fields

Field	Description
Name	A unique and recognizable name you provide for the partner link.
Process	Displays the BPEL process service component name.
WSDL URL	The name and location of the WSDL file or Java interface that you select for the partner link. Click the SOA Service Explorer icon (second icon from the left above the WSDL URL field) to access a window for selecting the WSDL file or Java interface to use. Java interfaces display for selection under the References folder with a name of javaEJB. If the component with which you are wiring this partner link uses WSDL files and you select a Java interface and click OK, a message displays indicating that this component requires a WSDL interface. If you click Yes, a compatible WSDL file is created based on the Java interface. For more information about integrating components that use Java interfaces into SOA composite applications, see Chapter 49, "Integrating the Spring Framework in SOA Composite Applications."
Partner Link Type	The partner link defined in the WSDL file.
Partner Role	The role performed by the partner link.
My Role	The role performed by the BPEL process service component. In this case, the BPEL process service component does not have a role because it is a synchronous process.

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

Best Practice: As a best practice, always create and wire Oracle Mediator and BPEL process service components in the SOA Composite Editor, instead of in Oracle BPEL Designer.

If you add an Oracle Mediator or BPEL process partner link to your BPEL process in Oracle BPEL Designer and connect either partner link to your BPEL process through an invoke activity, the wiring is not automatically reflected above in the SOA Composite Editor. You must explicitly wire the Oracle Mediator or BPEL process service component to your BPEL process again in the SOA Composite Editor.

Note that this is not an issue with human task or business rule partner links in Oracle BPEL Designer; both are also automatically wired in the SOA Composite Editor.

4.4 Creating a Partner Link

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

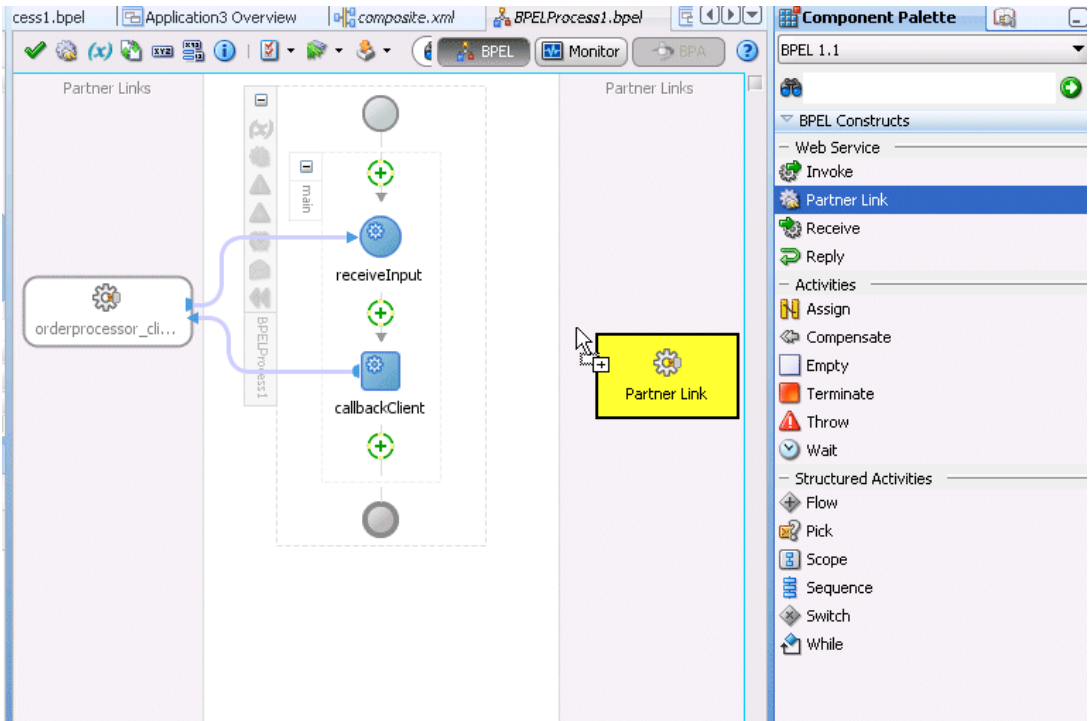
Likewise, creating and wiring a service or reference binding component to a BPEL process service component in the SOA Composite Editor causes a partner link to display in Oracle BPEL Designer.

4.4.1 How to Create a Partner Link

To create a partner link:

1. In the SOA Composite Editor, double-click the BPEL process service component. Oracle BPEL Designer is displayed.
2. In the Component Palette, expand **BPEL Constructs**.
3. Drag a **Partner Link** into the appropriate **Partner Links** swimlane, as shown in [Figure 4–10](#).

Figure 4–10 Partner Link Creation in Oracle BPEL Designer



The Create Partner Link dialog appears.

- 4. Complete the fields for this dialog, as described in Table 4–3.

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

4.4.1.1 Partner Links for an Outbound Adapter

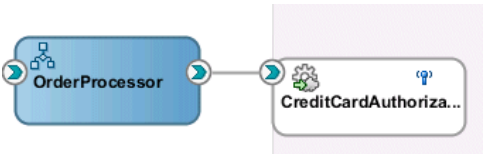
Table 4–4 describes the impact on the SOA Composite Editor.

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

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A partner link for an <i>outbound</i> adapter	■ 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 4–11 shows how this method of creation appears in the SOA Composite Editor.

Figure 4–11 SOA Composite Editor Impact



4.4.1.2 Partner Links for an Inbound Adapter

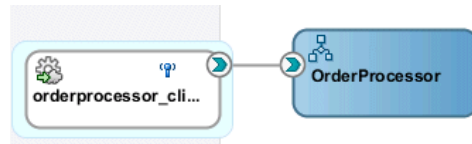
Table 4–5 describes the impact on the SOA Composite Editor.

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

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A partner link for an <i>inbound</i> adapter	■ 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 4–12 shows how this method of creation appears in the SOA Composite Editor.

Figure 4–12 SOA Composite Editor Impact



4.4.1.3 Partner Links from an Abstract WSDL to Call a Service

Table 4–6 describes the impact on the SOA Composite Editor.

Table 4–6 Impact of Partner Link Creation on the SOA Composite Editor

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

4.4.1.4 Partner Links from an Abstract WSDL to Implement a Service

Table 4–7 describes the impact on the SOA Composite Editor.

Table 4–7 Impact of Partner Link Creation on the SOA Composite Editor

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

Figure 4–13 shows how this method of creation appears in the SOA Composite Editor.

Figure 4–13 SOA Composite Editor Impact

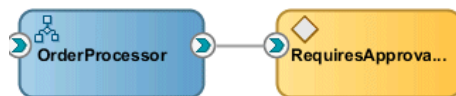
4.4.1.5 Partner Links and Human Tasks or Business Rules

Table 4–8 describes the impact on the SOA Composite Editor.

Table 4–8 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A human task or business rule is created	■ A human task or business rule in the composite ■ A reference for the BPEL service component ■ A wire connecting the BPEL service component to the new human task or business rule

Figure 4–14 shows how this method of creation appears in the SOA Composite Editor.

Figure 4–14 SOA Composite Editor Impact

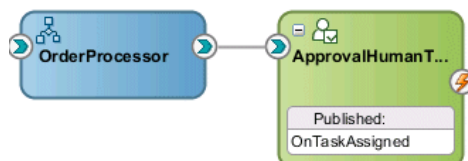
4.4.1.6 Partner Links from an Existing Human Task, Business Rule, or Oracle Mediator

Table 4–9 describes the impact on the SOA Composite Editor.

Table 4–9 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A partner link by dragging an existing human task, business rule, or mediator service component into the BPEL process	■ A reference for the BPEL service component ■ A wire connecting the BPEL service component to the existing human task, business rule, or mediator

Figure 4–15 shows how this method of creation appears in the SOA Composite Editor.

Figure 4–15 SOA Composite Editor Impact

4.5 Introduction to Technology Adapters

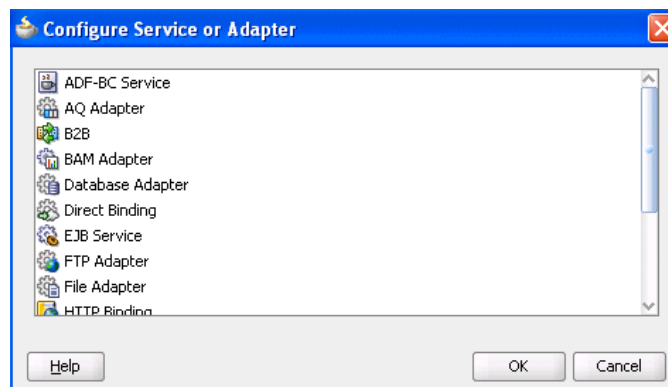
The Partner Link dialog shown in [Figure 4–9](#) also enables you to take advantage of another key feature that Oracle BPEL Process Manager and Oracle JDeveloper provide. Click the **Service Wizard** icon shown in [Figure 4–16](#) to access the Adapter Configuration wizard.

Figure 4–16 Defining an Adapter



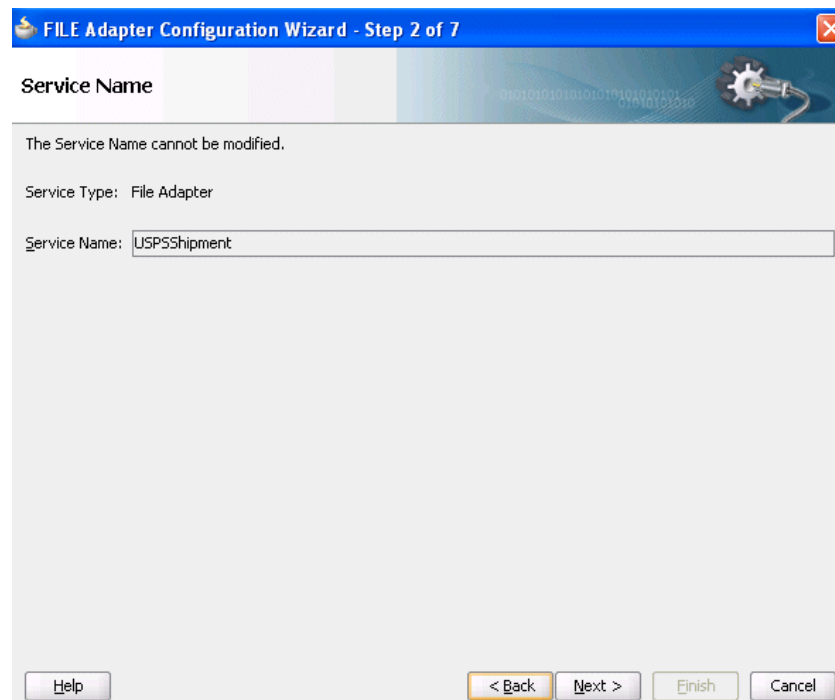
Adapters enable you to integrate the BPEL process service component (and, therefore, the SOA composite application as a whole) with access to file systems, FTP servers, database tables, database queues, sockets, Java Message Services (JMS), and MQ. You can also integrate with services such as HTTP binding, direct binding, EJB, and others. This wizard enables you to configure the types of services and adapters shown in [Figure 4–17](#) for use with the BPEL process service component:

Figure 4–17 Service and Adapter Types



For information about the service and adapter types, see [Chapter 34, "Getting Started with Binding Components."](#)

When you select an adapter type, the Service Name window shown in [Figure 4–18](#) prompts you to enter a name. For this example, **File Adapter** was selected in [Figure 4–17](#). When the wizard completes, a WSDL file by this service name appears in the Application Navigator for the BPEL process service component (for this example, named **USPSShipment.wsdl**). The service name must be unique within the project. This file includes the adapter configuration settings you specify with this wizard. Other configuration files (such as header files and files specific to the adapter) are also created and display in the Application Navigator.

Figure 4–18 Adapter Service Name

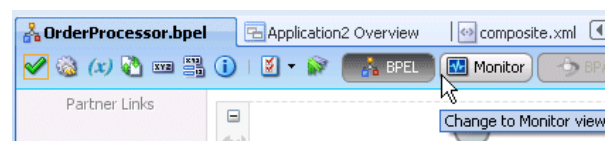
The Adapter Configuration wizard windows that appear after the Service Name window are based on the adapter type you selected.

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

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

4.6 Introduction to BPEL Process Monitors

You can configure BPEL process monitors in Oracle BPEL Designer by selecting **Monitor** at the top of Oracle BPEL Designer. [Figure 4–19](#) provides details. BPEL process monitors can send data to Oracle BAM for analysis and graphical display through the Oracle BAM adapter.

Figure 4–19 BPEL Process Monitors

For more information, see [Section 50.3, "Using Oracle BAM Monitor Express With BPEL Processes."](#)

Introduction to Interaction Patterns in a BPEL Process

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

This chapter includes the following sections:

- [Section 5.1, "Introduction to One-Way Messages"](#)
- [Section 5.2, "Introduction to Synchronous Interactions"](#)
- [Section 5.3, "Introduction to Asynchronous Interactions"](#)
- [Section 5.4, "Introduction to Asynchronous Interactions with a Timeout"](#)
- [Section 5.5, "Introduction to Asynchronous Interactions with a Notification Timer"](#)
- [Section 5.6, "Introduction to One Request, Multiple Responses"](#)
- [Section 5.7, "Introduction to One Request, One of Two Possible Responses"](#)
- [Section 5.8, "Introduction to One Request, a Mandatory Response, and an Optional Response"](#)
- [Section 5.9, "Introduction to Partial Processing"](#)
- [Section 5.10, "Introduction to Multiple Application Interactions"](#)

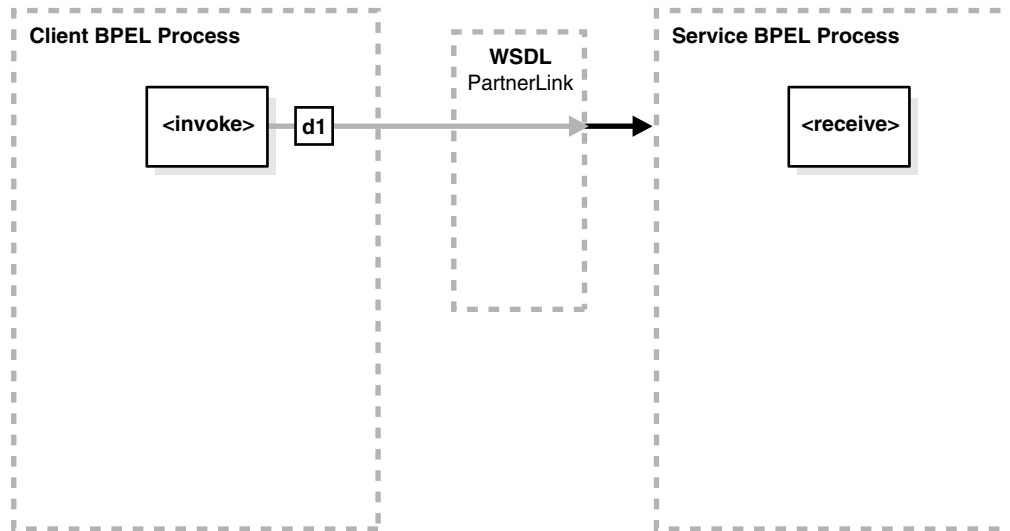
5.1 Introduction to One-Way Messages

In a one-way message, or fire and forget, the client sends a message to the service (d1 in [Figure 5-1](#)), and the service is not required to reply. The client sending the message does not wait for a response, but continues executing immediately. [Example 5-1](#) shows the `portType` and `operation` part of the BPEL process WSDL file for this environment.

Example 5-1 One-Way WSDL File

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

[Figure 5-1](#) provides an overview.

Figure 5–1 One-Way Message**BPEL Process Service Component as the Client**

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

BPEL Process Service Component as the Service

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

5.2 Introduction to Synchronous Interactions

In a synchronous interaction, a client sends a request to a service (d1 in [Figure 5–2](#)), and receives an immediate reply (d2 in [Figure 5–2](#)). A BPEL process service component can be at either end of this interaction, and must be coded based on its role as either the client or the service. For example, a user requests a subscription to an online newspaper and immediately receives email confirmation that their request has been accepted. [Example 5–2](#) shows the portType and operation part of the BPEL process WSDL file for this environment.

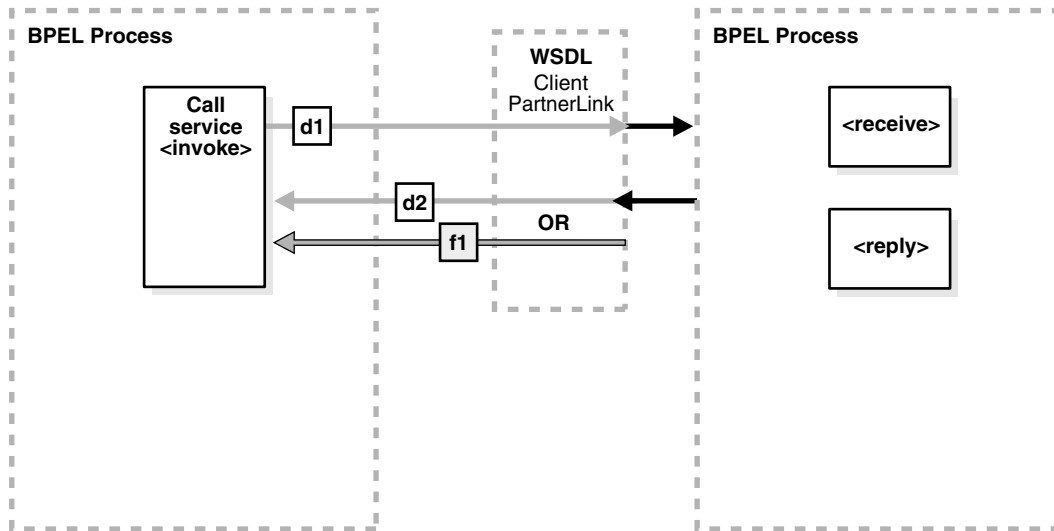
Example 5–2 Synchronous WSDL File

```

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

```

[Figure 5–2](#) provides an overview.

Figure 5–2 Synchronous Interaction**BPEL Process Service Component as the Client**

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

BPEL Process Service Component as the Service

When the BPEL process service component is on the service side of a synchronous transaction, it needs a receive activity to accept the incoming request, and a reply activity to return either the requested information or an error message (a fault; f1 in [Figure 5–2](#)) defined in the WSDL.

For more information about synchronous interactions, see [Chapter 7, "Invoking a Synchronous Web Service from a BPEL Process."](#)

5.3 Introduction to Asynchronous Interactions

In an asynchronous interaction, a client sends a request to a service and waits until the service replies. [Example 5–3](#) shows the `portType` and `operation` part of the BPEL process WSDL file for this environment.

Example 5–3 Asynchronous WSDL File

```

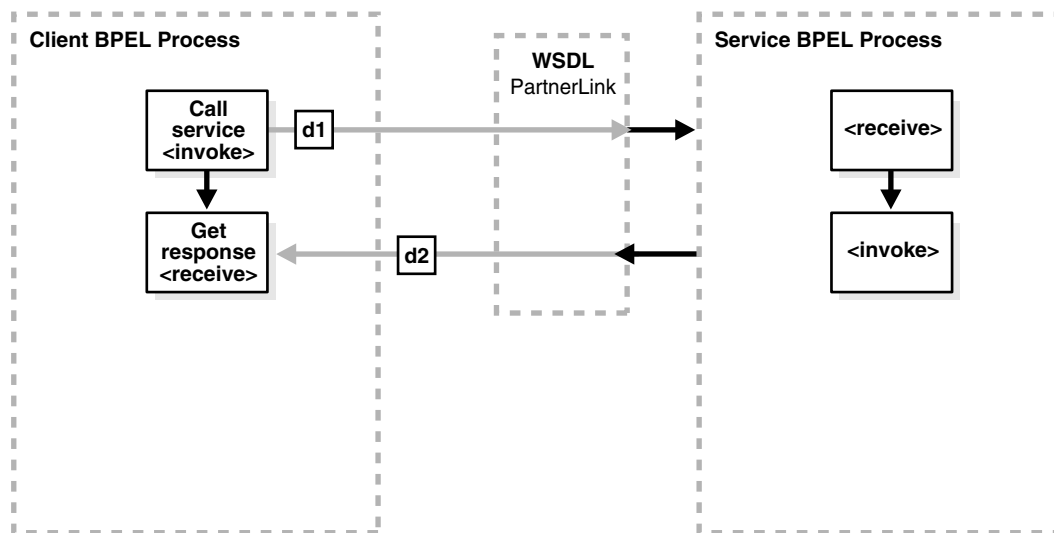
...
<wsdl:portType name="BPELProcess1">
  <wsdl:operation name="process">
    <wsdl:input message="client:BPELProcess1RequestMessage" />
  </wsdl:operation>
</wsdl:portType>

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

```

Figure 5–3 provides an overview.

Figure 5–3 Asynchronous Interaction



BPEL Process Service Component as the Client

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

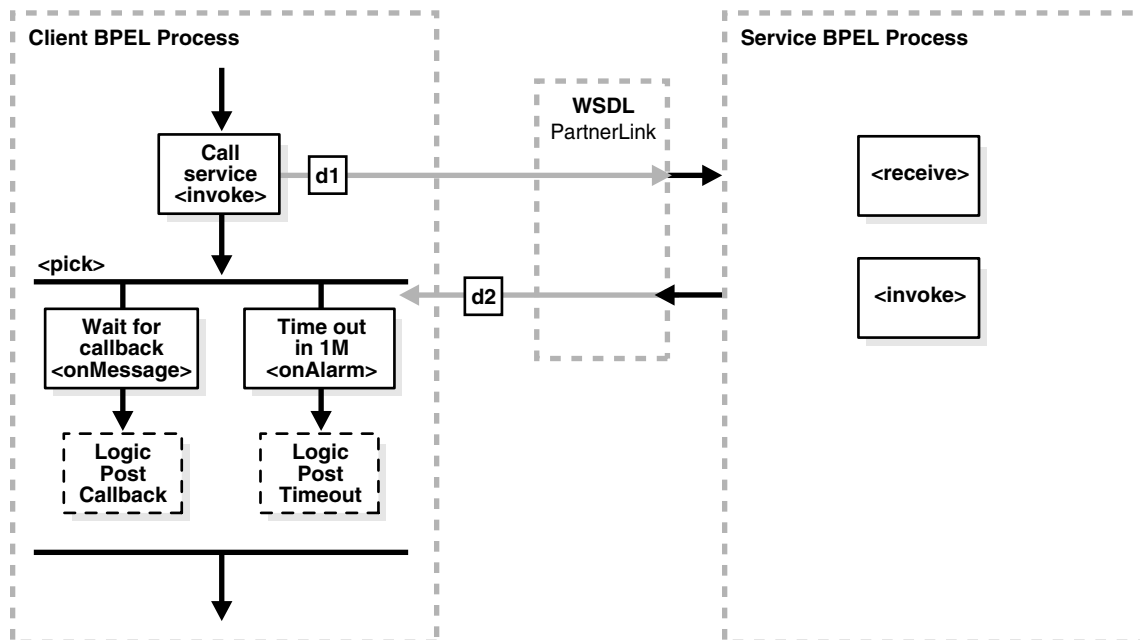
BPEL Process Service Component as the Service

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

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

5.4 Introduction to Asynchronous Interactions with a Timeout

In an asynchronous interaction with a timeout (which you perform in BPEL with a pick activity), a client sends a request to a service and waits until it receives a reply, or until a certain time limit is reached, whichever comes first. For example, a client requests a loan offer. If the client does not receive a loan offer reply within a specified amount of time, the request is canceled. [Figure 5–4](#) provides an overview.

Figure 5–4 Asynchronous Interaction with Timeout**BPEL Process Service Component as the Client**

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

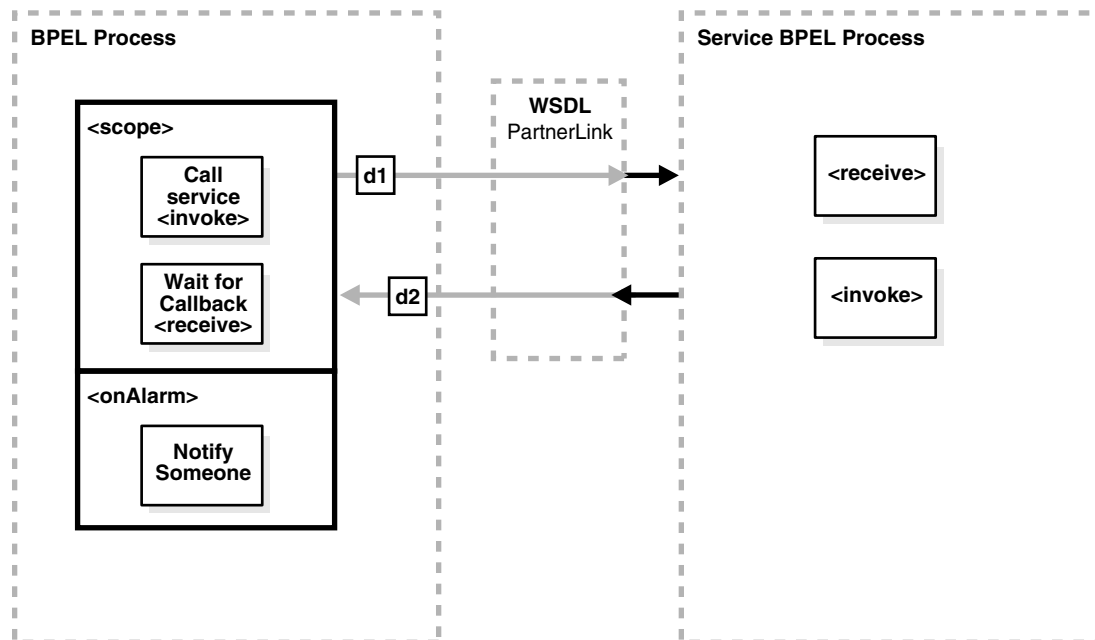
For more information about asynchronous interactions with a timeout, see [Section 14.2, "Creating a Pick Activity to Select Between Continuing a Process or Waiting."](#)

BPEL Process Service Component as the Service

The behavior of the BPEL process service component as a service matches the behavior with the asynchronous interaction with the BPEL process service component as the service.

5.5 Introduction to Asynchronous Interactions with a Notification Timer

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

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

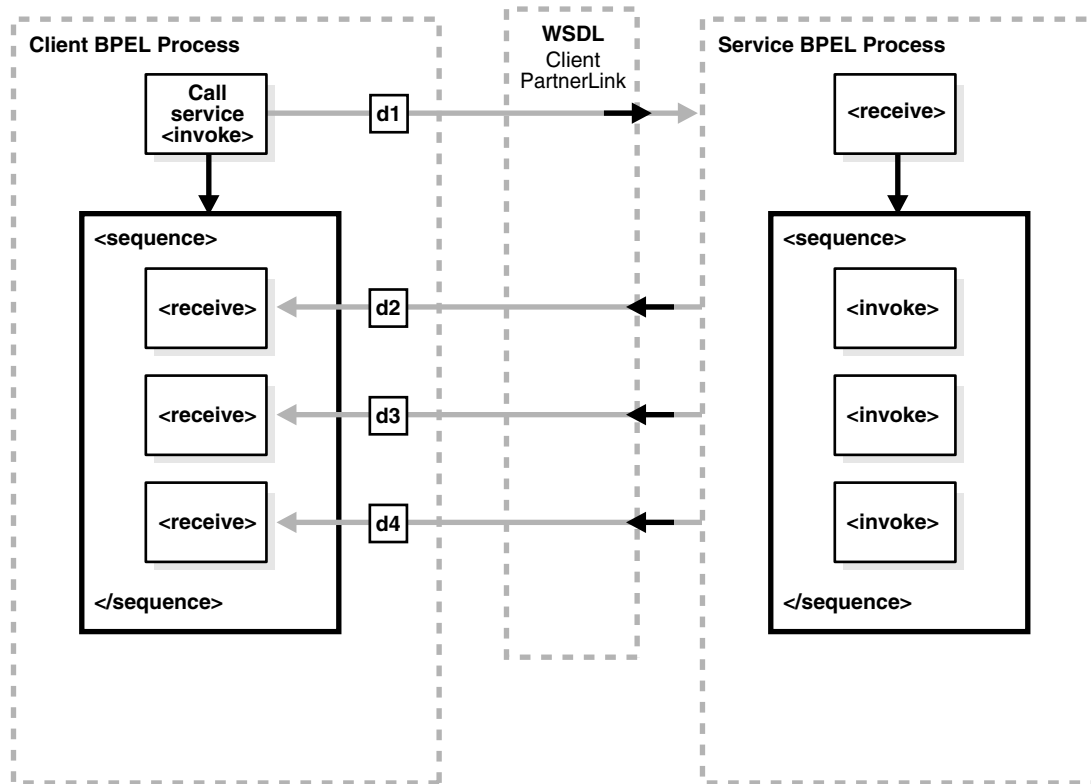
When the BPEL process service component is on the client side of this transaction, it needs a scope activity containing an invoke activity to send the request, and a receive activity to accept the reply. The onAlarm handler of the scope activity has a time limit and instructions on what to do when the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

BPEL Process Service Component as the Service

The behavior for the BPEL process service component as the service matches the behavior with the asynchronous interaction with the BPEL process service component as the service.

5.6 Introduction to One Request, Multiple Responses

In this interaction type, the client sends a single request to a service and receives multiple responses in return. For example, the request can be to order a product online, and the first response can be the estimated delivery time, the second response a payment confirmation, and the third response a notification that the product has shipped. In this example, the number and types of responses are expected. [Figure 5–6](#) provides an overview.

Figure 5–6 One Request, Multiple Responses**BPEL Process Service Component as the Client**

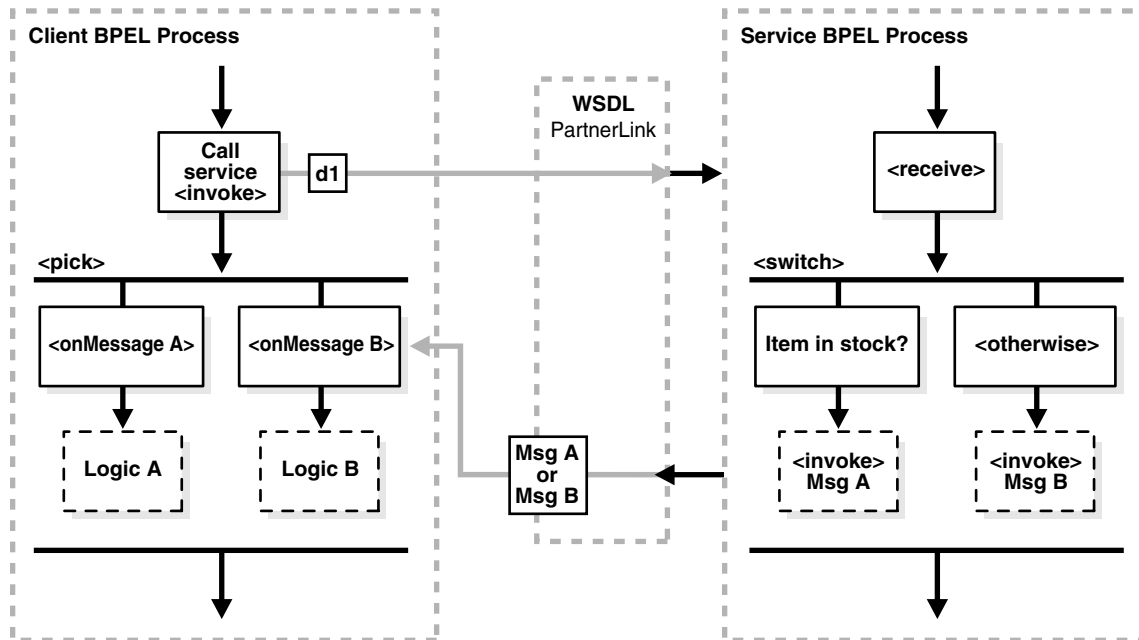
When the BPEL process service component is on the client side of this transaction, it needs an invoke activity to send the request, and a sequence activity with three receive activities, one for each reply. As with all partner activities, the WSDL file defines the interaction.

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.

5.7 Introduction to One Request, One of Two Possible Responses

In an interaction using one request and one of two possible responses, the client sends a single request to a service and receives one of two possible responses. For example, the request can be to order a product online, and the first response can be either an in-stock message or an out-of-stock message. [Figure 5–7](#) provides an overview.

Figure 5–7 One Request, One of Two Possible Responses**BPEL Process Service Component as the Client**

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

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

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

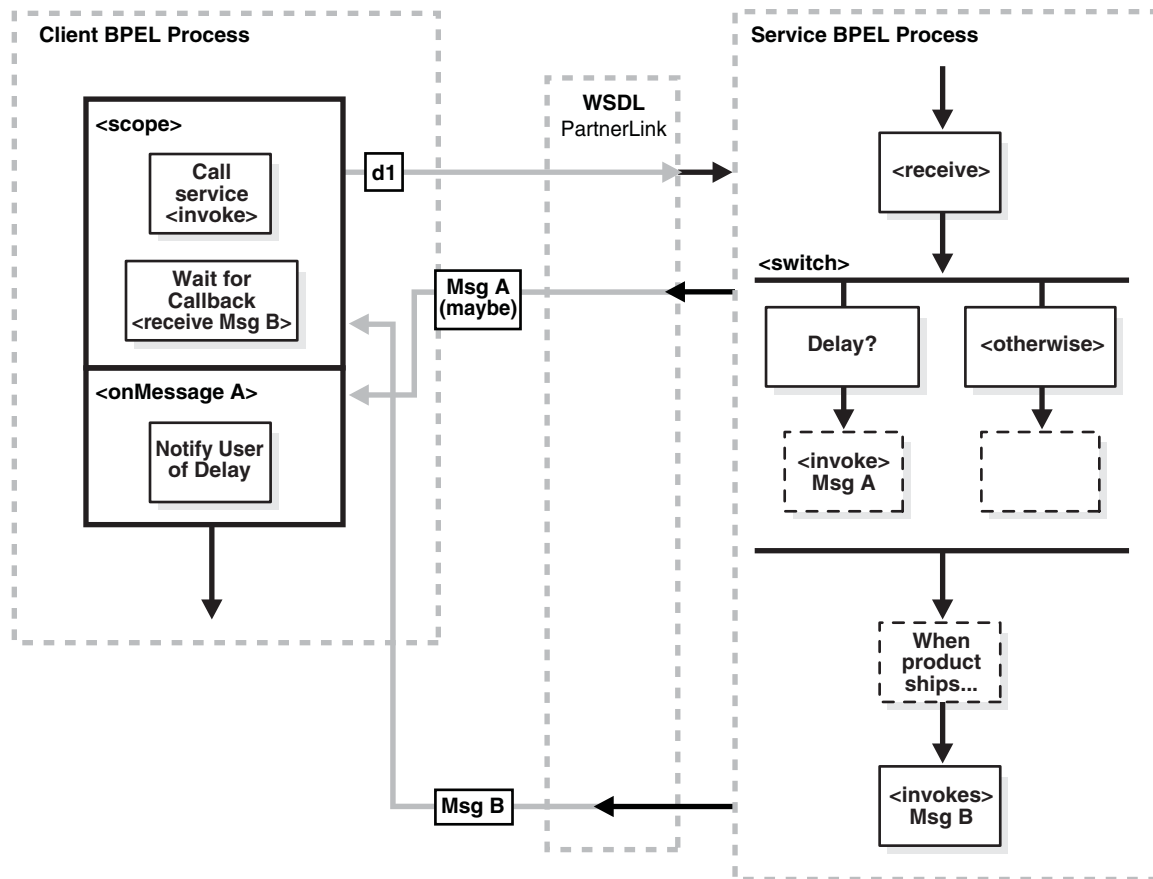
For more information about interactions using one request and one of two possible responses, see [Section 14.2, "Creating a Pick Activity to Select Between Continuing a Process or Waiting."](#)

BPEL Process Service Component as the Service

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

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

In this type of interaction, the client sends a single request to a service and receives one or two responses. Here, the request is to order a product online. If the product is delayed, the service sends a message letting the customer know. In any case, the service always sends a notification when the item ships. [Figure 5–8](#) provides an overview.

Figure 5–8 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

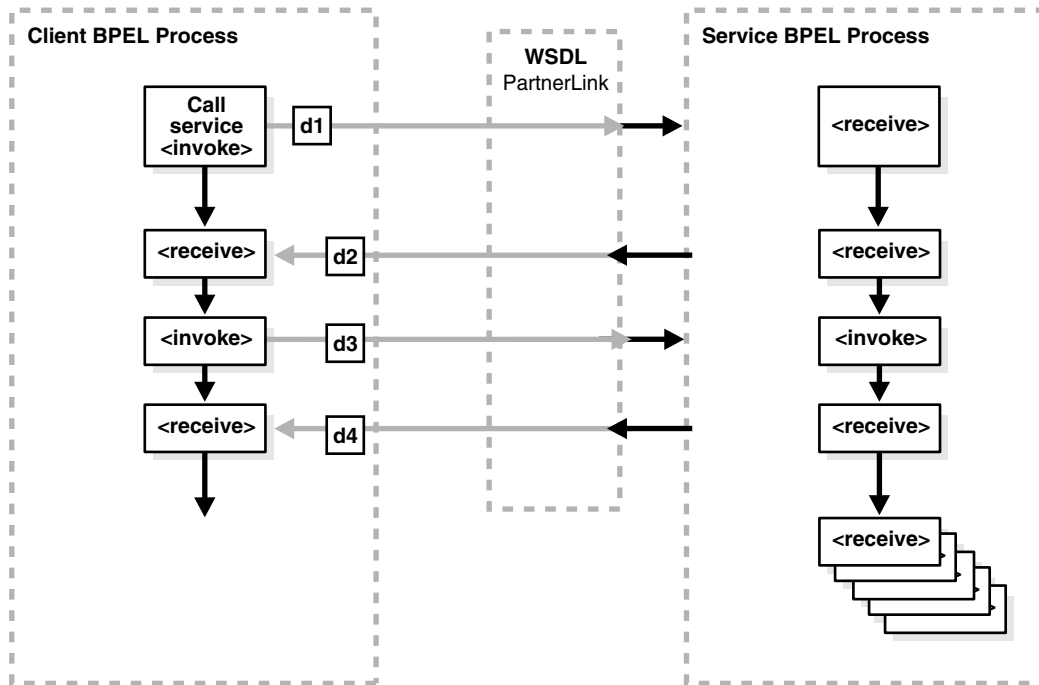
The BPEL service needs a scope activity containing the receive activity and an invoke activity to send the mandatory shipping message, and the scope's onAlarm handler to send the optional delayed message if a timer expires (for example, send the delayed message if the item is not shipped in 24 hours).

5.9 Introduction to Partial Processing

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

and so on. This pattern can also include multiple shot callbacks, followed by longer-term processing. [Figure 5-9](#) provides an overview.

Figure 5-9 Partial Processing



BPEL Process Service Component as the Client

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

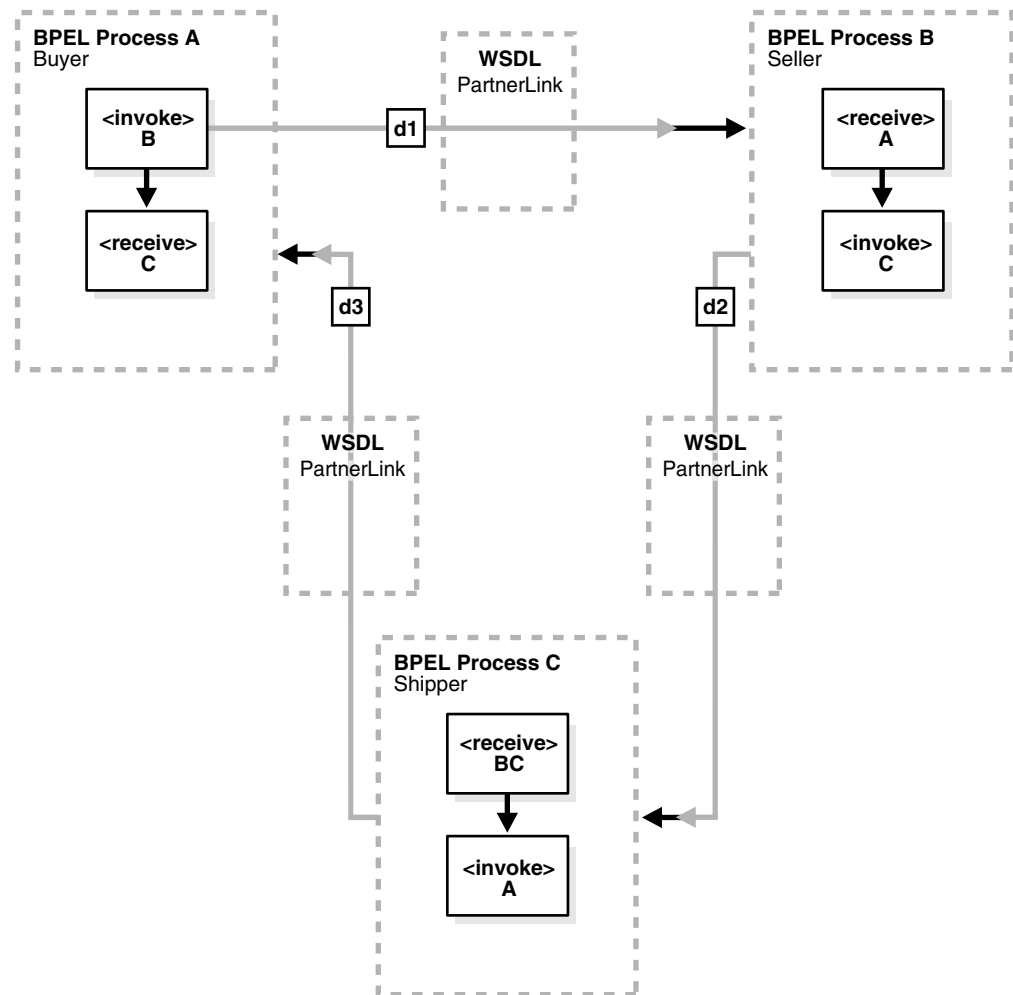
BPEL Process Service Component as the Service

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

5.10 Introduction to Multiple Application Interactions

In some cases, there are more than two applications involved in a transaction, for example, a buyer, seller, and shipper. In this case, the buyer sends a request to the seller, the seller sends a request to the shipper, and the shipper sends a notification to the buyer. This A-to-B-to-C-to-A transaction pattern can handle many transactions at the same time. Therefore, a mechanism is required for keeping track of which message goes where. [Figure 5-10](#) provides an overview.

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

Figure 5–10 Multiple Party Interactions

This kind of coordination can be managed using WS-Addressing or correlation sets. For more information about both, see [Chapter 8, "Invoking an Asynchronous Web Service from a BPEL Process."](#)

Manipulating XML Data in a BPEL Process

This chapter describes how to manipulate XML data in a BPEL process service component. This chapter provides a variety of examples. Topics include how to work with variables, sequences, and arrays; use XPath expressions; and perform tasks such as mathematical calculations. Supported specifications are also referenced.

This chapter includes the following sections:

- [Section 6.1, "Introduction to Manipulating XML Data in BPEL Processes"](#)
- [Section 6.2, "Delegating XML Data Operations to Data Provider Services"](#)
- [Section 6.3, "Using Standalone SDO-based Variables"](#)
- [Section 6.4, "Initializing a Variable with Expression Constants or Literal XML"](#)
- [Section 6.5, "Copying Between Variables"](#)
- [Section 6.6, "Accessing Fields in Element and Message Type Variables"](#)
- [Section 6.7, "Assigning Numeric Values"](#)
- [Section 6.8, "Using Mathematical Calculations with XPath Standards"](#)
- [Section 6.9, "Assigning String Literals"](#)
- [Section 6.10, "Concatenating Strings"](#)
- [Section 6.11, "Assigning Boolean Values"](#)
- [Section 6.12, "Assigning a Date or Time"](#)
- [Section 6.13, "Manipulating Attributes"](#)
- [Section 6.14, "Manipulating XML Data with bpelx Extensions"](#)
- [Section 6.15, "Validating XML Data"](#)
- [Section 6.16, "Using Element Variables in Message Exchange Activities in BPEL 2.0"](#)
- [Section 6.17, "Mapping WSDL Message Parts in BPEL 2.0"](#)
- [Section 6.18, "Importing Process Definitions in BPEL 2.0"](#)
- [Section 6.19, "Manipulating XML Data Sequences That Resemble Arrays"](#)
- [Section 6.20, "Converting from a String to an XML Element"](#)
- [Section 6.21, "Understanding Document-Style and RPC-Style WSDL Differences"](#)
- [Section 6.22, "Manipulating SOAP Headers in BPEL"](#)
- [Section 6.23, "Declaring Extension Namespaces in BPEL 2.0"](#)

Note: Most of the examples in this chapter assume that the WSDL file defining the associated message types is document-literal style rather than the RPC style. There is a difference in how XPath query strings are formed for RPC-style WSDL definitions. If you are working with a type defined in an RPC WSDL file, see [Section 6.21, "Understanding Document-Style and RPC-Style WSDL Differences."](#)

For Oracle BPEL Process Manager samples, visit the following URL:

<https://soasamples.samplecode.oracle.com>

6.1 Introduction to Manipulating XML Data in BPEL Processes

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

6.1.1 XML Data in BPEL

In a BPEL process service component, most pieces of data are in XML format. This includes the messages passed to and from the BPEL process service component, the messages exchanged with external services, and the local variables used by the process. You define the types for these messages and variables with the XML schema, usually in one of the following:

- Web Services Description Language (WSDL) file for the flow
- WSDL files for the services it invokes
- XSD file referenced by those WSDL files

Therefore, most variables in BPEL are XML data, and any BPEL process service component uses much of its code to manipulate these XML variables. This typically includes performing data transformation between representations required for different services, and local manipulation of data (for example, to combine the results from several service invocations).

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

6.1.2 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 6-1](#) shows the formal syntax for BPEL version 1.1, as described in the *Business Process Execution Language for Web Services Specification Version 1.1*:

Example 6-1 Assign Activity for BPEL 1.1

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

[Example 6-2](#) shows the formal syntax for BPEL version 2.0, as described in the *Web Services Business Process Execution Language Specification Version 2.0*. The `keepSrcElementName` attribute specifies whether the element name of the destination (as selected by the `to-spec`) is replaced by the element name of the source (as selected by the `from-spec`) during the copy operation. When `keepSrcElementName` is set to `no` (the default value), the name (that is, the namespace name and local name properties) of the original destination element is used as the name of the resulting element. When `keepSrcElementName` is set to `yes`, the source element name is used as the name of the resulting destination element.

Example 6-2 Assign Activity for BPEL 2.0

```
<assign validate="yes|no"? standard-attributes>
  standard-elements
  (
    <copy keepSrcElementName="yes|no"? ignoreMissingFromData="yes|no"?>
      from-spec
      to-spec
    </copy>
    . . .
    . . .
  )
</assign>
```

This syntax is described in detail in both specifications. The `from-spec` and `to-spec` typically specify a variable or variable part, as shown in [Example 6-3](#):

Example 6-3 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 Rules dialog that includes a **From** section and a **To** section. This reflects the preceding BPEL source code syntax.

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

- XPath queries

An XPath query selects a field within a source or target variable part. The `from` or `to` clause can include a query attribute whose value is an XPath query string.

[Example 6-4](#) provides an example:

Example 6-4 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 Version 1.1* (section 14.3) or *Web Services Business Process Execution Language Specification Version 2.0* (section 8.4), 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.

- For BPEL 1.1, 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 Version 1.1*. For more information about `getVariableData`, see [Section B.2.60.2, "getVariableData."](#)

- For BPEL 2.0, the extensions are also defined in the standard BPEL namespace <http://schemas.xmlsoap.org/ws/2003/03/business-process/>. However, the prefix is `bpel`:

```
<from>bpel:getVariableProperty('input', 'propertyName')</from>
```


For more information, see section 8.3 of the *Web Services Business Process Execution Language Specification Version 2.0*. For more information about `getVariableProperty`, see [Section B.2.60.4, "getVariableProperty \(For BPEL 2.0\)."](#)

- Oracle BPEL XPath extension functions

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

These functions are defined in the namespace

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

- Custom functions

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

Sophisticated data manipulation can be difficult to perform with the BPEL assign activity and the core XPath functions. However, you can perform complex data manipulation and transformation by using XSLT, Java, or a `bpelx` operation under an assign activity (See [Section 6.14, "Manipulating XML Data with bpelx Extensions"](#)) or as a web service. For XSLT, Oracle BPEL Process Manager includes XPath functions that execute these transformations.

For more information about XPath and XQuery transformation code examples, see [Chapter 37, "Creating Transformations with the XSLT Mapper."](#)

Note: Passing large schemas through an assign activity can cause Oracle JDeveloper to freeze up and run low on memory if you right-click the target or source payload node in the Edit Assign dialog and select **Expand All Child Nodes**. As a workaround, manually expand the payload elements.

6.2 Delegating XML Data Operations to Data Provider Services

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

Note: This feature is only supported in BPEL 1.1 projects.

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

In releases before 11g, variables and messages exchanged within a BPEL business process were a 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. Optionally performing the extra business data logic after certain operations, but before web service invocation, was sometimes preferred.

To address these challenges with Release 11g, 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 6–1](#) 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 6–1 Data Manipulation Capabilities in Previous and Current Releases

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

Note: Only BPEL process service components currently allow the use of SDO-formed variables. If your composite application has an Oracle Mediator service component wired with an SDO-based Java binding component reference, the data form of the variable defaults to DOM. In addition, the features described for 10.1.x releases in [Table 6–1](#) are still supported in release 11g.

The WebLogic Fusion Order Demo application describes use of the entity variable.

6.2.1 How to Create an Entity Variable

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

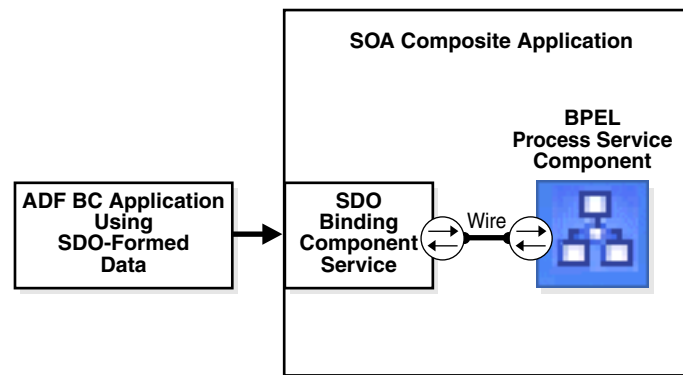
In 10.1.x releases of Oracle BPEL Process Manager, all variable data was in DOM form. With release 11g, variable data in SDO form is also supported. DOM and SDO variables in BPEL process service components are implicitly converted to the required forms. For example, an Oracle BPEL process service component using DOM-based variables can automatically convert these variables as required to SDO-based variables in an assign activity, and vice versa. Both form types are defined in the XSD schema file. No user intervention is required.

Entity variables also support SDO-formed data. However, unlike the DOM and SDO variables, the entity variable with SDO-based data enables you to bind a unique key value to data (for example, a purchase order). Only the key is stored in the dehydration store; the data requiring conversion is stored with the service of the Oracle ADF Business Component application. The key points to the data stored in the service. When the data is required, it is fetched from the data provider service and placed into memory. The process occurs in two places: the bind entity activity and the dehydration store. For example, when Oracle BPEL Process Manager rehydrates, it stores only the key for the entity variable; when it wakes up, it does an implicit bind to get the current data.

6.2.1.1 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 6–1](#).

Figure 6–1 Inbound Direction



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

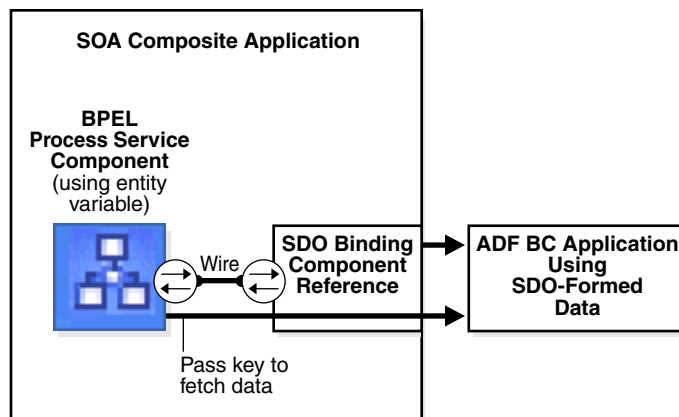
- Define an SDO binding component service and a BPEL process service component in the composite application.
- Connect (wire) the SDO service and BPEL process service component.
- Define the details of the BPEL process service component.

For more information about using the SOA Composite Editor, see [Chapter 2, "Developing SOA Composite Applications with Oracle SOA Suite."](#)

6.2.1.2 Understanding How SDO Works in the Outbound Direction

The SDO binding component reference enables messages to be sent from the composite application to Oracle ADF Business Component application external partners in the outside world, as shown in [Figure 6–2](#).

Figure 6–2 Outbound Direction



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

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

6.2.1.3 Creating an Entity Variable and Choosing a Partner Link

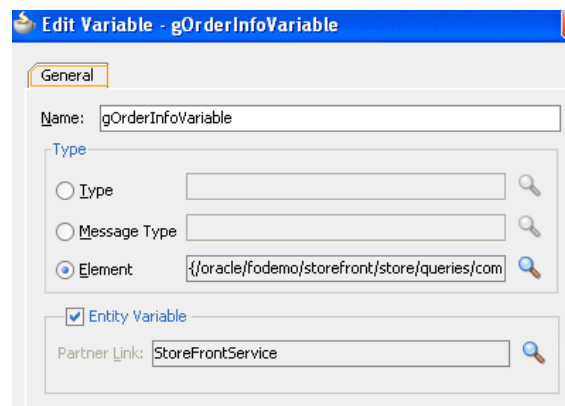
You now create an entity variable and select a partner link for the Oracle ADF Business Component application. The following example describes how the OrderProcessor BPEL process service component receives an ID for an order by using a bind entity activity to point to order data in an Oracle ADF Business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

To create an entity variable and choose a partner link:

1. Go to the Structure window of the BPEL process service component in Oracle JDeveloper.

2. Right-click the **Variables** folder and select **Expand All Child Nodes**.
3. In the second **Variables** folder, right-click and select **Create Variable**.
The Create Variable dialog appears.
4. In the **Name** field, enter a name.
5. Click the **Entity Variable** checkbox and select the **Search** icon to the right of the **Partner Link** field.
The Partner Link Chooser dialog appears with a list of available services, including the SDO service called **ADF-BC Service**.
6. Browse for and select the service for the Oracle ADF Business Component application.
7. Click **OK** to close the Partner Link Chooser and Create Variable dialogs.
The dialog looks as shown in [Figure 6–3](#).

Figure 6–3 Create Variable Dialog



6.2.1.4 Creating a Binding Key

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

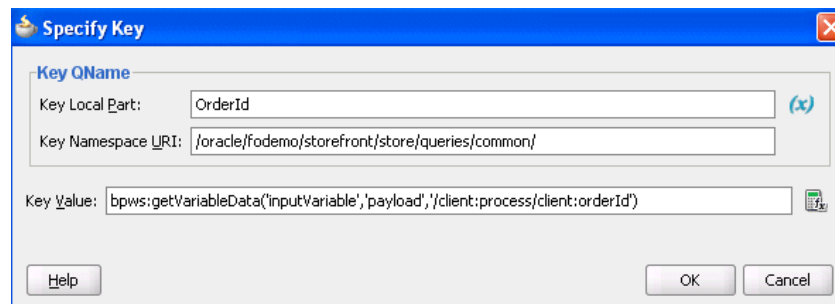
To create a binding key:

1. In the Component Palette for a BPEL 1.1 project, expand **Oracle Extensions**.
2. Drag a **Bind Entity** activity into your BPEL process service component.
The Bind Entity dialog appears.
3. In the **Name** field, enter a name.
4. To the right of the **Entity Variable** field, click the **Search** icon.
The Variable Chooser dialog appears.
5. Select the entity variable created in [Section 6.2.1.3, "Creating an Entity Variable and Choosing a Partner Link"](#) and click **OK**.
6. In the **Unique Keys** section, click the **Add** icon.
The Specify Key dialog appears. You use this dialog to create a key for retrieving the order ID from the Oracle ADF Business Component data provider service.
7. Enter the details described in [Table 6–2](#) to define the binding key:

Table 6–2 Specify Key Dialog Fields and Values

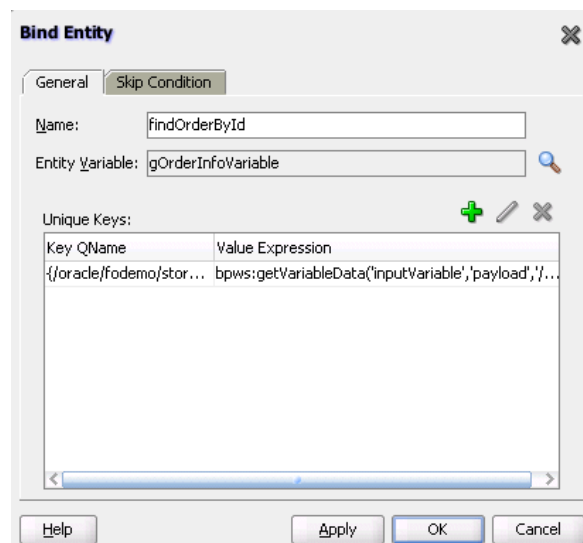
Field	Value
Key Local Part	Enter the local part of the key.
Key Namespace URI	Enter the namespace URI for the key.
Key Value	Enter the key value expression. This expression must match the type of a key. The following examples show expression value keys for a POID key: <code>\$inputMsg.payload/tns:poid</code> <code>bpws:getVariableData('inputmsg','payload','tns:poid')</code> 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.

Figure 6–4 shows the Specify Key dialog after completion.

Figure 6–4 Specify Key Dialog

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

A name-pair value appears in the **Unique Keys** table, as shown in Figure 6–5. Design is now complete.

Figure 6–5 Bind Entity Dialog

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

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

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

6.3 Using Standalone SDO-based Variables

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

6.3.1 How to Declare SDO-based Variables

The syntax for declaring an SDO-based variable is similar to that for declaring BPEL variables. [Example 6-5](#) provides details.

Example 6-5 SDO-based Variable Declaration

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

If you want to override the automatic detection, use the `bpelx:sdoCapable="true|false"` switch. For example, variable `deptVar_v` described in [Example 6-5](#) is a regular DOM-based variable. [Example 6-6](#) provides an example of the schema.

Example 6-6 XSD Sample

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

6.3.2 How to Convert from XML to SDO

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

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

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

Example 6-7 XML-to-SDO Conversion

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

[Example 6-8](#) provides an example of copying from an XPath expression of an SDO variable to a DOM variable.

Example 6–8 Copy from an XPath Expression of an SDO Variable to a DOM Variable

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

Example 6–9 provides an example of removing a portion of SDO data.

Example 6–9 SDO Data Removal

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

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

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

6.4 Initializing a Variable with Expression Constants or Literal XML

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

6.4.1 How To Assign a Literal XML Element

Example 6–10 assigns a literal `result` element to the payload part of the output variable:

Example 6–10 Literal Element Assignment

```

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

```

6.5 Copying Between Variables

When you copy between variables, you copy directly from one variable (or part) to another variable of a compatible type, without needing to specify a particular field within either variable. In other words, you do not need to specify an XPath query.

6.5.1 How to Copy Between Variables

[Example 6–11](#) 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 6–11 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 6–12](#):

Example 6–12 Variable Definition

```

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

```

The WSDL file defines the person message type shown in [Example 6–13](#):

Example 6–13 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 Version 1.1*. For BPEL 2.0, see Section 8.4.4 of *Web Services Business Process Execution Language Specification Version 2.0* for a similar example.

For more information, see [Section A.2.2, "Assign Activity."](#)

6.5.2 Initializing Variables with an Inline from-spec in BPEL 2.0

A variable can optionally be initialized by using an inline `from-spec`. Click the **Initialize** tab in the Create Variable dialog in a BPEL 2.0 project to create this type of variable.

Inline variable initializations are conceptually designed as a virtual sequence activity that includes a series of virtual assign activities, one for each variable being initialized, in the order in which they appear in the variable declarations. Each virtual assign activity contains a single virtual copy operation whose `from-spec` is as given in the variable initialization. The `to-spec` points to the variable being created.

[Example 6-14](#) provides details.

Example 6-14 Variable Initialization with an Inline from-spec

```
<variables>
  <variable name="tmp" element="tns:output">
    <from>
      <literal>
        <output xmlns="http://samples.otn.com/bpel2.0/ch8.1">
          <value>1000</value>
        </output>
      </literal>
    </from>
  </variable>
</variables>
```

For more information, see section 8.1 of *Web Services Business Process Execution Language Specification Version 2.0*.

6.6 Accessing Fields in Element and Message Type Variables

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

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

In [Example 6-15](#), 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 6-15 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 6–16](#) shows how the BPEL file defines message type-based variables involved in this assignment:

Example 6–16 BPEL File Definition - Message Type-Based Variables in BPEL 1.1

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

The `crInput` variable is used as an input message to a credit rating service. Its message type, `CreditFlowRequestMessage`, is defined in the `CreditFlowService.wsdl` file, as shown in [Example 6–17](#):

Example 6–17 CreditFlowRequestMessage Definition

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

`CreditFlowRequest` is defined with a field named `ssn`. The message type `CreditRatingServiceRequestMessage` is defined in the `CreditRatingService.wsdl` file, as shown in [Example 6–18](#):

Example 6–18 CreditRatingServiceRequestMessage Definition

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

[Example 6–19](#) shows the BPEL 2.0 syntax for how the BPEL file defines message type-based variables involved in the assignment in [Example 6–15](#). Note that `/tns:CreditFlowRequest` is not required.

Example 6–19 BPEL File Definition - Message Type-Based Variables in BPEL 2.0

```
<copy>
  <from>$input.payload/tns:ssn</from>
  <to>$crInput.payload</to>
</copy>
```

A BPEL process can also use element-based variables. [Example 6–20](#) shows how to use element-based variables in BPEL 1.1. The `autoloan` field is copied from the loan application process's input message into the `customer` field of a web service's input message.

Example 6–20 Field Copying Levels in BPEL 1.1

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

[Example 6–21](#) shows how to use element-based variables in BPEL 2.0.

Example 6–21 Field Copying Levels in BPEL 2.0

```

<assign>
  <copy>
    <from>$input.payload/autoloan:application/autoloan:customer</from>
    <to>$customer</to>
  </copy>
</assign>

```

[Example 6–22](#) shows how the BPEL file defines element-based variables involved in an assignment:

Example 6–22 BPEL File Definition - Element-Based Variables

```

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

```

6.7 Assigning Numeric Values

You can assign numeric values in XPath expressions.

6.7.1 How to Assign Numeric Values

[Example 6–23](#) shows how to assign an XPath expression with the integer value of 100.

Example 6–23 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>

```

6.8 Using Mathematical Calculations with XPath Standards

You can use simple mathematical expressions like the one in [Section 6.8.1, "How To Use Mathematical Calculations with XPath Standards,"](#) which increment a numeric value.

6.8.1 How To Use Mathematical Calculations with XPath Standards

In [Example 6–24](#), 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 6–24 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 in BPEL 1.1, as shown in [Example 6–25](#):

Example 6–25 \$variable Syntax Use in BPEL 1.1

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

[Example 6–26](#) shows how to use \$variable syntax in BPEL 2.0.

Example 6–26 \$variable Syntax Use in BPEL 2.0

```
<assign>
  <copy>
    <from>$input.payload + 1</from>
    <to>$output.payload</to>
  </copy>
</assign>
```

6.9 Assigning String Literals

You can assign string literals to a variable in BPEL.

6.9.1 How to Assign String Literals

The code in [Example 6–27](#) copies a BPEL 1.1 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 6–27 Expression Copy in BPEL 1.1

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

[Example 6–28](#) shows how to perform this expression in BPEL 2.0.

Example 6–28 Expression Copy in BPEL 2.0

```
<assign>
  <copy>
    <from>'GE'</from>
    <to>$output.payload/p:symbol</from>
  </copy>
</assign>
```

6.10 Concatenating Strings

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

6.10.1 How to Concatenate Strings

The concatenation is accomplished with the core XPath function named `concat`; in addition, the variable value involved in the concatenation is retrieved with the BPEL XPath function `getVariableData`. In [Example 6–29](#), `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 6–29 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*.

6.11 Assigning Boolean Values

You can assign boolean values with the XPath boolean function.

6.11.1 How to Assign Boolean Values

[Example 6–30](#) provides an example of assigning boolean values in BPEL 1.1. 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 6–30 Boolean Value Assignment in BPEL 1.1

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

[Example 6–31](#) provides an example of assigning boolean values in BPEL 2.0.

Example 6–31 Boolean Value Assignment in BPEL 2.0

```
<assign>
  <copy>
    <from>true()</from>
    <to>$.output.payload/approved</to>
  </copy>
</assign>
```

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

If you instead use `"boolean(true)"` or `"boolean(false)"`, the `true` or `false` inside the boolean function is interpreted as a relative element step, and not as any `true` or `false` constant. It attempts to select a child node named `true` under the

current XPath context node. In most cases, the `true` node does not exist. Therefore, an empty result node set is returned and the `boolean()` function in XPath 1.0 converts an empty node set into a false result. This result can be potentially confusing.

6.12 Assigning a Date or Time

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

For related information, see section 9.1.2 of the *Business Process Execution Language for Web Services Specification Version 1.1* and section 8.3.2 of the *Web Services Business Process Execution Language Specification Version 2.0*.

6.12.1 How to Assign a Date or Time

[Example 6-32](#) shows an example that uses the function `getCurrentDate` in BPEL 1.1.

Example 6-32 Date or Time Assignment in BPEL 1.1

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

[Example 6-33](#) shows an example that uses the function `getCurrentDate` in BPEL 2.0.

Example 6-33 Date or Time Assignment in BPEL 2.0

```
<assign>
  <copy>
    <from>xpath20:getCurrentDate()</from>
    <to>$output.payload/invoiceDate</to>
  </copy>
</assign>
```

In [Example 6-34](#), 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 6-34 formatDate Function in BPEL 1.1

```
<!-- 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>
```


[Example 6–35](#) shows how the `formatDate` function works in BPEL 2.0.

Example 6–35 *formatDate Function in BPEL 2.0*

```
<assign>
  <copy>
    <from>ora:formatDate('2005-06-10T15:56:00','MMM dd, yyyy')</from>
    <to>$output.payload/formattedDate</to>
  </copy>
</assign>
```

6.13 Manipulating Attributes

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

6.13.1 How to Manipulate Attributes

The code in [Example 6–36](#) fetches and copies the `custId` attribute from this XML data:

Example 6–36 *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 BPEL 1.1 code in [Example 6–37](#) selects the `custId` attribute of the customer field and assigns it to the variable `custId`:

Example 6–37 *custId Attribute Select and Assign Operations in BPEL 1.1*

```
<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>
```

[Example 6–38](#) shows the equivalent syntax in BPEL 2.0 for selecting the `custId` attribute of the customer field and assigning it to the variable `custId`:

Example 6–38 *custId Attribute Select and Assign Operations in BPEL 2.0*

```
<assign>
  <copy>
    <from>$input.payload/autoloan:application/autoloan:customer/@custId</from>
```

```
<to>$custId</to>
</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 6–39](#):

Example 6–39 *custId* Attribute Definition

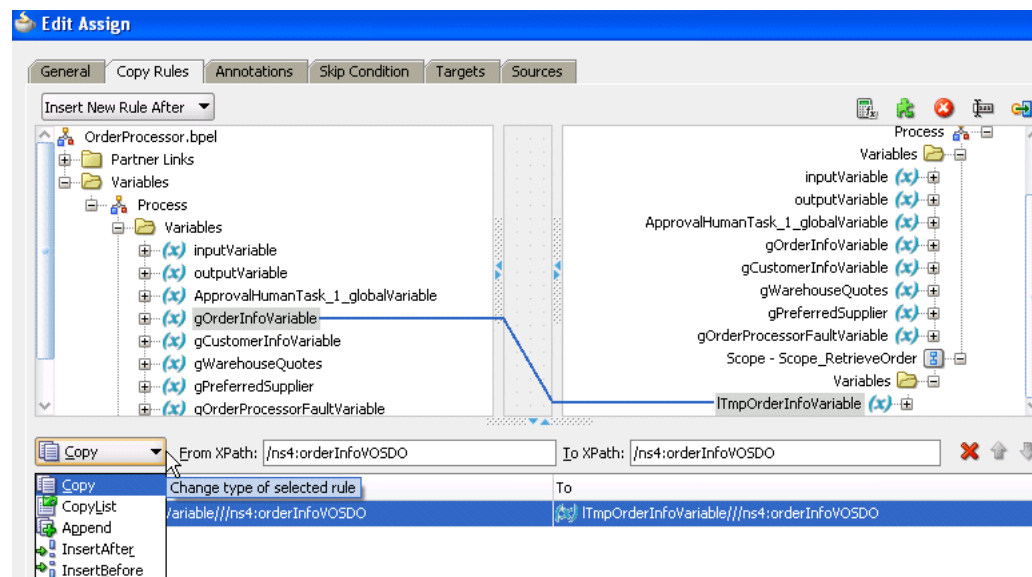
```
<complexType name="CustomerProfileType">
  <sequence>
    <element name="name" type="string"/>
    ...
  </sequence>
  <attribute name="custId" type="string"/>
</complexType>
```

6.14 Manipulating XML Data with bpelx Extensions

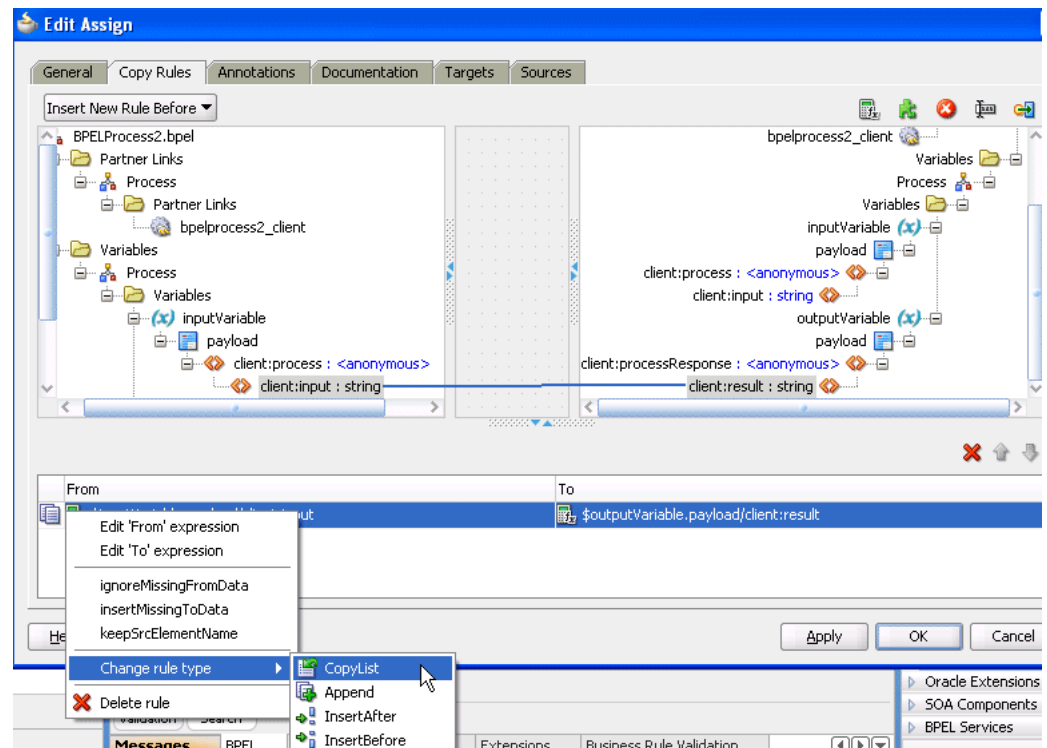
You can perform various operations on XML data in assign activities. The `bpelx` extension types described in this section provide this functionality. In Oracle BPEL Designer, you can add `bpelx` extension types at the bottom of the **Copy Rules** tab of an Assign dialog. After creating a copy rule, you select it and then choose a `bpelx` extension type from the dropdown list in BPEL 1.1 or the context menu in BPEL 2.0. This changes the copy rule to the selected extension type.

In BPEL 1.1, you select an extension type from the dropdown list, as shown in [Figure 6–6](#).

Figure 6–6 Copy Rule Converted to `bpelx` Extension in BPEL 1.1



In BPEL 2.0, you select an extension type by right-clicking the copy rule, selecting **Change rule type**, and then selecting the extension type, as shown in [Figure 6–7](#).

Figure 6–7 Copy Rule Converted to bpelx Extension in BPEL 2.0

For more information, see the online Help for this dialog and [Section A.2.2, "Assign Activity."](#)

6.14.1 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. To use this extension, you select a copy rule at the bottom of the **Copy Rules** tab, then select **Append** from the dropdown list, as shown in [Figure 6–6](#).

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

6.14.1.1 bpelx:append in BPEL 1.1

[Example 6–40](#) provides an example of `bpelx:append` in a BPEL project that supports BPEL version 1.1.

Example 6–40 bpelx:append Extension in BPEL 1.1

```
<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 6-41](#) consolidates multiple bills of material into one single bill of material (BOM) by appending multiple `b:parts` for one BOM to `b:parts` of the consolidated BOM.

Example 6-41 Consolidation of Multiple Bills of Material

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

6.14.1.2 bpelx:append in BPEL 2.0

[Example 6-42](#) provides an example of `bpelx:append` syntax in a BPEL project that supports BPEL version 2.0. In BPEL 2.0, the functionality is the same as described in [Section 6.14.1.1, "bpelx:append in BPEL 1.1,"](#) but the syntax is slightly different.

Example 6-42 bpelx:append Extension in BPEL 2.0

```
<bpel:assign>
  <bpelx:append>
    <bpelx:from>${billOfMaterialVar}/b:parts/b:part</bpelx:from>
    <bpelx:to>${consolidatedBillOfMaterialVar}/b:parts</bpelx:from>
  </bpelx:append>
</bpel:assign>
```

6.14.2 How to Use bpelx:insertBefore

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

The `bpelx:insertBefore` extension in an assign activity enables a BPEL process service component to insert the contents of one variable, expression, or XML fragment before another variable's contents. To use this extension, you select a copy rule at the bottom of the **Copy Rules** tab, then select **InsertBefore** from the dropdown list, as shown in [Figure 6-6](#).

6.14.2.1 bpelx:insertBefore in BPEL 1.1

[Example 6-43](#) provides an example of `bpelx:insertBefore` in a BPEL project that supports BPEL version 1.1.

Example 6-43 bpelx:insertBefore Extension in BPEL 1.1

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

```

    </bpelx:insertBefore>
  </bpel:assign>

```

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

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

[Example 6-44](#) shows the syntax before the execution of `<insertBefore>`. The value of `addrVar` is:

Example 6-44 Presyntax Execution

```

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

```

[Example 6-45](#) shows the syntax after the execution:

Example 6-45 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 6-46](#) shows the value of `addrVar`:

Example 6-46 addrVar Value

```

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

```

6.14.2.2 bpelx:insertBefore in BPEL 2.0

[Example 6-47](#) provides an example of `bpelx:insertBefore` syntax in a BPEL project that supports BPEL version 2.0. In BPEL 2.0, the functionality is the same as described in [Section 6.14.2.1, "bpelx:insertBefore in BPEL 1.1,"](#) but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:insertBefore` extension.

Example 6-47 bpelx:insertBefore Extension in BPEL 2.0

```

<assign>
  <extensionAssignOperation>
    <bpelx:insertBefore>

```

```
<bpelx:from>
  <bpelx:literal>
    <a:city>Redwood Shore</a:city>
  </bpelx:literal>
</bpelx:from>
<bpelx:to>$addrVar/a:state</bpelx:to>
</bpelx:insertBefore>
</extensionAssignOperation>
</assign>
```

6.14.3 How to Use bpelx:insertAfter

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

The bpelx:insertAfter extension in an assign activity enables a BPEL process service component to insert the contents of one variable, expression, or XML fragment after another variable's contents. To use this extension, you select a copy rule at the bottom of the **Copy Rules** tab, then select **InsertAfter** from the dropdown list, as shown in [Figure 6-6](#).

6.14.3.1 bpelx:insertAfter in BPEL 1.1

[Example 6-48](#) provides an example of bpelx:insertAfter in a BPEL project that supports BPEL version 1.1.

Example 6-48 bpelx:insertAfter Extension in BPEL 1.1

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

This operation is similar to the functionality described for [Section 6.14.2, "How to Use bpelx:insertBefore,"](#) except for the following:

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

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

[Example 6-49](#) shows the syntax before the execution of <insertAfter>. The value of addrVar is:

Example 6-49 Presyntax Execution

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

[Example 6–50](#) shows the syntax after the execution:

Example 6–50 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 6–51](#) shows the value of addrVar:

Example 6–51 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.

6.14.3.2 bpelx:insertAfter in BPEL 2.0

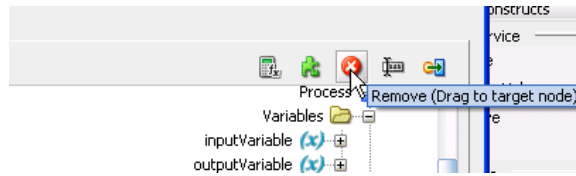
[Example 6–52](#) provides an example of `bpelx:insertAfter` syntax in a BPEL project that supports BPEL version 2.0. In BPEL 2.0, the functionality is the same as described in [Section 6.14.3.1, "bpelx:insertAfter in BPEL 1.1,"](#) but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:insertAfter` extension.

Example 6–52 bpelx:insertAfter Extension in BPEL 2.0

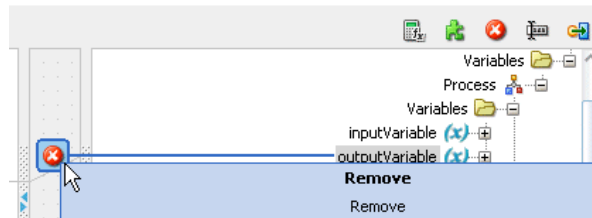
```
<assign>
  <extensionAssignOperation>
    <bpelx:insertAfter>
      <bpelx:from>
        <bpelx:literal>
          <a:addressLine>Mailstop 1op6</a:addressLine>
        </bpelx:literal>
      </bpelx:from>
      <bpelx:to>$addrVar/a:addressLine[1]</bpelx:to>
    </bpelx:insertAfter>
  </extensionAssignOperation>
</assign>
```

6.14.4 How to Use bpelx:remove

The `bpelx:remove` extension in an assign activity enables a BPEL process service component to remove a variable. In Oracle BPEL Designer, you add the `bpelx:remove` extension by dragging the **remove** icon in the upper right corner of the **Copy Rules** tab to the target variable you want to remove, and releasing the cursor. [Figure 6–8](#) provides details.

Figure 6–8 Remove Icon in Copy Rules Tab of an Assign Activity

After releasing the cursor, the `bpelx:remove` extension is applied to the target variable. Figure 6–9 provides details.

Figure 6–9 `bpelx:remove` Extension Applied to a Target Variable

6.14.4.1 `bpelx:remove` in BPEL 1.1

Example 6–53 provides an example of `bpelx:remove` in a BPEL project that supports BPEL version 1.1.

Example 6–53 `bpelx:remove` Extension in BPEL 1.1

```
<bpel:assign>
  <bpelx:remove>
    <bpelx:target variable="ncname" part="ncname"? query="xpath_str" />
  </bpelx:remove>
</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 6–54 shows `addrVar` with the following value:

Example 6–54 `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 6–55 in the BPEL process service component file, the second address line of Mailstop is removed:

Example 6–55 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 6–56](#) in the BPEL process service component file, both address lines are removed:

Example 6–56 Removal of Both Address Lines

```

<bpel:assign>
  <bpelx:remove>
    <target variable="addrVar"
      query="/a:usAddress/a:addressLine" />
  </bpelx:remove>
</bpel:assign>

```

6.14.4.2 bpelx:remove in BPEL 2.0

[Example 6–57](#) provides an example of `bpelx:remove` syntax in a BPEL project that supports BPEL version 2.0. In BPEL 2.0, the functionality is the same as described in [Section 6.14.4.1, "bpelx:remove in BPEL 1.1,"](#) but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:remove`.

Example 6–57 bpelx:remove Extension in BPEL 2.0

```

<assign>
  <extensionAssignOperation>
    <bpelx:remove>
      <bpelx:target>$ncname.ncname/xpath_str</bpelx:target>
    </bpelx:remove>
  </extensionAssignOperation>
</assign>

```

6.14.5 How to Use bpelx:rename and XSD Type Casting

The `bpelx:rename` extension in an assign activity enables a BPEL process service component to rename an element through use of XSD type casting. In Oracle BPEL Designer, you add the `bpelx:rename` extension by dragging the **rename** icon in the upper right corner of the **Copy Rules** tab to the target variable you want to remove, and releasing the cursor. The rename icon displays to the right of the **remove** icon shown in [Figure 6–8](#). After releasing the cursor, the Rename dialog is displayed for renaming the target variable.

6.14.5.1 bpelx:rename in BPEL 1.1

[Example 6–58](#) provides an example of `bpelx:rename` in a BPEL project that supports BPEL version 1.1.

Example 6–58 bpelx:rename Extension in BPEL 1.1

```

<bpel:assign>
  <bpelx:rename elementTo="QName1"? typeCastTo="QName2"?>
    <bpelx:target variable="ncname" part="ncname"? query="xpath_str" />
  </bpelx:rename>
</bpel:assign>

```

The syntax of `bpelx:target` is similar to and a subset of `to-spec` for the copy operation. The target must return a list of one more element nodes. Otherwise, a `bpel:selectionFailure` fault is generated. The element nodes specified in the `from-spec` are renamed to the `QName` specified by the `elementTo` attribute. The `xsi:type` attribute is added to those element nodes to cast those elements to the `QName` type specified by the `typeCastTo` attribute.

Assume you have the employee list shown in [Example 6–59](#):

Example 6–59 *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 6–60](#):

Example 6–60 *Application of Promotion Changes*

```
<bpel:assign>
  <bpelx:rename typeCastTo="e:ManagerType">
    <bpelx:target variable="emplistVar"
      query="/e:emplist/e:emp[./e:firstName='Peter' and
./e:lastName='Smith']" />
  </bpelx:rename>
</bpel:assign>
```

After executing the above casting (renaming), the data looks as shown in [Example 6–61](#) with `xsi:type` info added to Peter Smith:

Example 6–61 *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 6-62](#) provides an example.

Example 6-62 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 6-63](#):

Example 6-63 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>

```

6.14.5.2 bpelx:rename in BPEL 2.0

[Example 6-64](#) provides an example of `bpelx:rename` syntax in a BPEL project that supports BPEL version 2.0. In BPEL 2.0, the functionality is the same as described in [Section 6.14.5.1, "bpelx:rename in BPEL 1.1,"](#) but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:rename`.

Example 6-64 bpelx:rename Extension in BPEL 2.0

```

<bpel:assign>
  <extensionAssignOperation>
    <bpelx:rename elementTo="QName1"? typeCastTo="QName2"?>
      <bpelx:target>$ncname[.ncname][./xpath_str]</bpelx:target>
    </bpelx:rename>
  </extensionAssignOperation>
</bpel:assign>

```

6.14.6 How to Use bpelx:copyList

The `bpelx:copyList` extension in an `assign` activity enables a BPEL process service component to perform a `copyList` operation of the contents of one variable, expression, or XML fragment to another variable.

To use this extension in BPEL 1.1, you select a copy rule at the bottom of the **Copy Rules** tab, then select **copyList** from the dropdown list, as shown in [Figure 6-6](#). To use this extension in BPEL 2.0, you right-click a copy rule, select **Change rule type**, and select **CopyList**, as shown in [Figure 6-7](#).

6.14.6.1 bpelx:copyList in BPEL 1.1

[Example 6-65](#) provides an example of `bpelx:copyList` in a BPEL project that supports BPEL version 1.1.

Example 6-65 *bpelx:copyList Extension in BPEL 1.1*

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

The `from-spec` query can yield a list of either all attribute nodes or all element nodes. The `to-spec` query can yield a list of L-value nodes: either all attribute nodes or all element nodes.

All the element nodes returned by the `to-spec` query must have the same parent element. If the `to-spec` query returns a list of element nodes, all element nodes must be contiguous.

If the `from-spec` query returns attribute nodes, then the `to-spec` query must return attribute nodes. Likewise, if the `from-spec` query returns element nodes, then the `to-spec` query must return element nodes. Otherwise, a `bpws:mismatchedAssignmentFailure` fault is thrown.

The `from-spec` query can return zero nodes, while the `to-spec` query must return at least one node. If the `from-spec` query returns zero nodes, the effect of the `copyList` operation is similar to the `remove` operation.

The `copylist` operation provides the following features:

- Removes all the nodes pointed to by the `to-spec` query.
- If the `to-spec` query returns a list of element nodes and there are leftover child nodes after removal of those nodes, the nodes returned by the `from-spec` query are inserted before the next sibling of the last element specified by the `to-spec` query. If there are no leftover child nodes, an `append` operation is performed.
- If the `to-spec` query returns a list of attribute nodes, those attributes are removed from the parent element. The attributes returned by the `from-spec` query are then appended to the parent element.

For example, assume a schema is defined as shown in [Example 6-66](#).

Example 6-66 *Schema*

```
<schema attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://xmlns.oracle.com/Event_jws/Event/EventTest"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="process">
    <complexType>
      <sequence>
        <element name="payload" type="string"
          maxOccurs="unbounded" />
      </sequence>
    </complexType>
  </element>
</schema>
```

```

        </sequence>
      </complexType>
    </element>
    <element name="processResponse">
      <complexType>
        <sequence>
          <element name="payload" type="string"
            maxOccurs="unbounded" />
        </sequence>
      </complexType>
    </element>
  </schema>

```

The `from` variable contains the content shown in [Example 6–67](#).

Example 6–67 Variable Content

```

<ns1:process xmlns:ns1="http://xmlns.oracle.com/Event_jws/Event/EventTest">
  <ns1: payload >a</ns1: payload >
  <ns1: payload >b</ns1: payload >
</ns1:process>

```

The `to` variable contains the content shown in [Example 6–68](#).

Example 6–68 Variable Content

```

<ns1:processResponse xmlns:ns1="http://xmlns.oracle.com/Event_
jws/Event/EventTest">
  <ns1: payload >c</ns1: payload >
</ns1:process>

```

The `bpelx:copyList` operation looks as shown in [Example 6–69](#).

Example 6–69 bpelx:copyList

```

<assign>
  <bpelx:copyList>
    <bpelx:from variable="inputVariable" part="payload"
      query="/client:process/client:payload"/>
    <bpelx:to variable="outputVariable" part="payload"
      query="/client:processResponse/client:payload"/>
  </bpelx:copyList>
</assign>

```

This makes the `to` variable as shown in [Example 6–70](#).

Example 6–70 Variable Content

```

<ns1:processResponse xmlns:ns1="http://xmlns.oracle.com/Event_
jws/Event/EventTest">
  <ns1: payload >a</ns1: payload >
  <ns1: payload >b</ns1: payload >
</ns1:process>

```

6.14.6.2 bpelx:copyList in BPEL 2.0

[Example 6–71](#) provides an example of `bpelx:copyList` syntax in a BPEL project that supports BPEL version 2.0. In BPEL 2.0, the functionality is the same as described in [Section 6.14.6.1, "bpelx:copyList in BPEL 1.1,"](#) but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:copyList` extension.

Example 6–71 bpelx:copyList Extension in BPEL 2.0

```

<assign>
  <extensionAssignOperation>
    <bpelx:copyList>
      <bpelx:from>${inputVariable.payload/client:payload}</bpelx:from>
      <bpelx:to>${outputVariable.payload/client:payload}</bpelx:to>
    </bpelx:copyList>
  </extensionAssignOperation>
</assign>

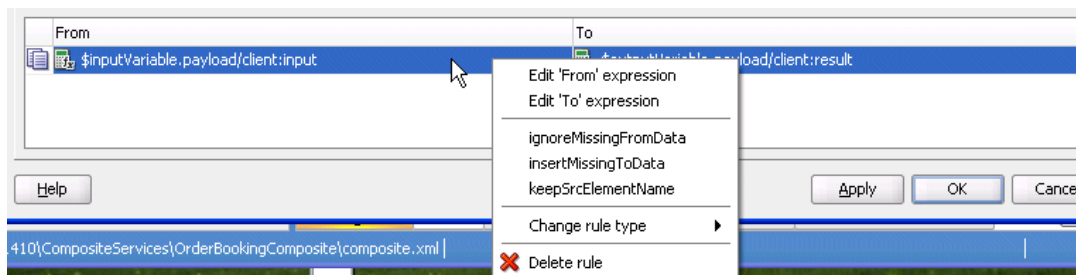
```

6.14.7 How to Use Assign Extension Attributes

You can assign the following attributes to copy rules in an assign activity.

- ignoreMissingFromData
- insertMissingToData
- keepSrcElementName

At the bottom of the **Copy Rules** tab of an assign activity, you right-click a selected copy rule to display a menu for choosing the appropriate attribute. [Figure 6–10](#) provides details.

Figure 6–10 Assign Extension Attributes**6.14.7.1 ignoreMissingFromData Attribute**

The ignoreMissingFromData attribute suppresses any `bpel:selectionFailure` standard faults. [Table 6–3](#) describes the syntax differences between BPEL versions 1.1 and 2.0.

Table 6–3 ignoreMissingFromData Attribute Syntax

BPEL 1.1	BPEL 2.0
<pre><copy bpelx:ignoreMissingFromData="yes no"/></pre>	<pre><copy ignoreMissingFromData="yes no"/></pre>

6.14.7.2 insertMissingToData Attribute

The insertMissingToData attribute instructs runtime to complete the (XPath) L-value specified by the `to-spec`, if no items were selected. [Table 6–4](#) describes the syntax differences between BPEL versions 1.1 and 2.0.

Table 6–4 insertMissingToData Attribute Syntax

BPEL 1.1	BPEL 2.0
<pre><copy bpelx:insertMissingToData="yes no"/></pre>	<pre><copy bpelx:insertMissingToData="yes no"/></pre>

6.14.7.3 keepSrcElementName Attribute

The `keepSrcElementName` attribute enables you to replace the element name of the destination (as selected by the `to-spec`) with the element name of the source. This attribute was not implemented in BPEL 1.1. [Table 6–5](#) describes the syntax supported in BPEL version 2.0.

Table 6–5 *keepSrcElementName Attribute Syntax*

BPEL 1.1	BPEL 2.0
Not implemented	<code><copy keepSrcElementName="yes no"/></code>

6.15 Validating XML Data

You can verify code and identify invalid XML data in a BPEL project.

6.15.1 How to Validate XML Data in BPEL 1.1

- In an assign activity in Oracle BPEL Designer:
 1. From the **BPEL Constructs** section of the Component Palette, drag an **Assign** activity into the designer.
 2. Double-click the **Assign** activity.
 3. In the **General** tab, enter a name for the activity and select the **Validate** checkbox.
 4. Click **Apply**, then **OK**.
 5. Click the **Source** tab to view the syntax.

```
<assign name=Assign1" bpelx:validate="yes"
. . .
</assign>
```

- In a standalone, extended validate activity in Oracle BPEL Designer that can be used without an assign activity:
 1. From the **Oracle Extensions** section of the Component Palette, drag a **Validate** activity into the designer.
 2. Double-click the **Validate** icon.
 3. Enter a name for the activity.
 4. Click the **Add** icon to select the variable to validate.
 5. Select the variable, then click **OK**.
 6. Click **Apply**, then **OK**.
 7. Click the **Source** tab to view the syntax.

```
<bpelx:validate name=Validate1" variables="inputVariable"/>
```

6.15.2 How to Validate XML Data in BPEL 2.0

- In an assign activity in Oracle BPEL Designer:
 1. From the **BPEL Constructs** section of the Component Palette, drag an **Assign** activity into the designer.
 2. Double-click the **Assign** activity.

3. In the **General** tab, enter a name for the activity and select the **Validate** checkbox.
4. Click **Apply**, then **OK**.
5. Click the **Source** tab to view the syntax. Note that the syntax for validating XML data with the assign activity is slightly different between BPEL versions 1.1 and 2.0.

```
<assign name="Assign1" validate="yes">
  . . .
</assign>
```

- In a standalone, extended validate activity in Oracle BPEL Designer that can be used without an assign activity:
 1. From the **BPEL Constructs** section of the Component Palette, drag a **Validate** activity into the designer.
 2. Double-click the **Validate** icon.
 3. Enter a name for the activity.
 4. Click the **Add** icon to select the variable to validate.
 5. Select the variable, then click **OK**.
 6. Click **Apply**, then **OK**.
 7. Click the **Source** tab to view the syntax. Note that the syntax for validating XML data with the validate activity is slightly different between BPEL versions 1.1 and 2.0.

```
<validate name="Validate1" variables="inputVariable"/>
```

6.16 Using Element Variables in Message Exchange Activities in BPEL 2.0

You can specify variables in the following message exchange activities:

- The **Input** field (for an `inputVariable` attribute) and **Output** field (for an `outputVariable` attribute) of an `invoke` dialog
- The **Input** field (for a `variable` attribute) of a `receive` activity
- The **Output** field (for a `variable` attribute) of a `reply` activity

The variables referenced by these fields typically must be message type variables in which the QName matches the QName of the input and output message types used in the operation, respectively.

The one exception is if the WSDL operation in the activity uses a message containing exactly one part that is defined using an element. In this case, a variable of the same element type used to define the part can be referenced by the `inputVariable` and `outputVariable` attributes, respectively, in the `invoke` activity or the `variable` attribute of the `receive` or `reply` activity.

Using a variable in this situation must be the same as declaring an anonymous, temporary WSDL message variable based on the associated WSDL message type.

Copying element data between the anonymous, temporary WSDL message variable and the element variable acts as a single virtual assign with one copy operation whose `keepSrcElementName` attribute is set to `yes`. The virtual assign must follow the same rules and use the same faults as a real assign activity. [Table 6-6](#) provides details.

Table 6–6 Mapping WSDL Message Parts

For The...	The...
inputVariable attribute	Value of the variable referenced by the attribute sets the value of the part in the anonymous temporary WSDL message variable.
outputVariable attribute	Value of the received part in the temporary WSDL message variable sets the value of the variable referenced by the attribute.
Receive activity	Incoming part's value sets the value of the variable referenced by the variable attribute.
Reply activity	Value of the variable referenced by the variable attribute sets the value of the part in the anonymous, temporary WSDL message variable that is sent out. For a reply activity sending a fault, the same scenario applies.

For more information about the `keepSrcElementName` attribute, see [Section 6.14.7.3, "keepSrcElementName Attribute."](#)

6.17 Mapping WSDL Message Parts in BPEL 2.0

The `toParts` element in `invoke` and `reply` activities provides an alternative to explicitly creating multipart WSDL messages from the contents of BPEL variables.

When you use the `toParts` element, as shown in [Example 6–72](#), an anonymous, temporary WSDL variable is defined based on the type specified by the input message of the appropriate WSDL operation.

Example 6–72 `toParts` Element

```
<toParts>
  <toPart part="payload" fromVariable="request"/>
</toParts>
```

The `toParts` element acts as a single, virtual assign activity. Each `toPart` acts as a copy operation. One `toPart` at most exists for each part in the WSDL message definition. Each copy operation copies data from the variable specified in the `fromVariable` attribute into the part of the anonymous, temporary WSDL variable referenced in the `part` attribute of the `toParts` element.

The `fromParts` element in `receive` activities, `invoke` activities, the `onEvent` branch of `scope` activities, and the `onMessage` branch of `pick` activities is similar to the `toParts` element. The `fromParts` element, as shown in [Example 6–73](#), retrieves data from an incoming multipart WSDL message and places the data into individual variables.

Example 6–73 `fromParts` Element

```
<fromParts>
  <fromPart part="payload" toVariable="request"/>
</fromParts>
```

When a WSDL message is received on an `invoke` activity that uses `fromParts` elements, the message is placed in an anonymous, temporary WSDL variable of the type specified by the output message of the appropriate WSDL operation.

As with the `toParts` element, the `fromParts` element acts as a single virtual assign activity. Each `fromPart` acts as a copy operation. Each copy operation copies the data at the part of the anonymous, temporary WSDL variable referenced in the `part` attribute of the `fromPart` into the variable indicated in the `toVariable` attribute.

For both the `toParts` and `fromParts` elements, the virtual assign activity must follow the same semantics and generate the same faults as a real assign activity.

The presence of a `fromParts` element in an invoke activity does not require it to have a `fromPart` for every part in the WSDL message definition. Parts not explicitly represented by `fromParts` elements are not copied from the anonymous WSDL variable to the variable.

For more information about mapping WSDL message parts with the `toParts` and `fromParts` elements, see the *Web Services Business Process Execution Language Version 2.0 Specification* located at the following URL:

<http://www.oasis-open.org>

6.17.1 How to Map WSDL Message Parts

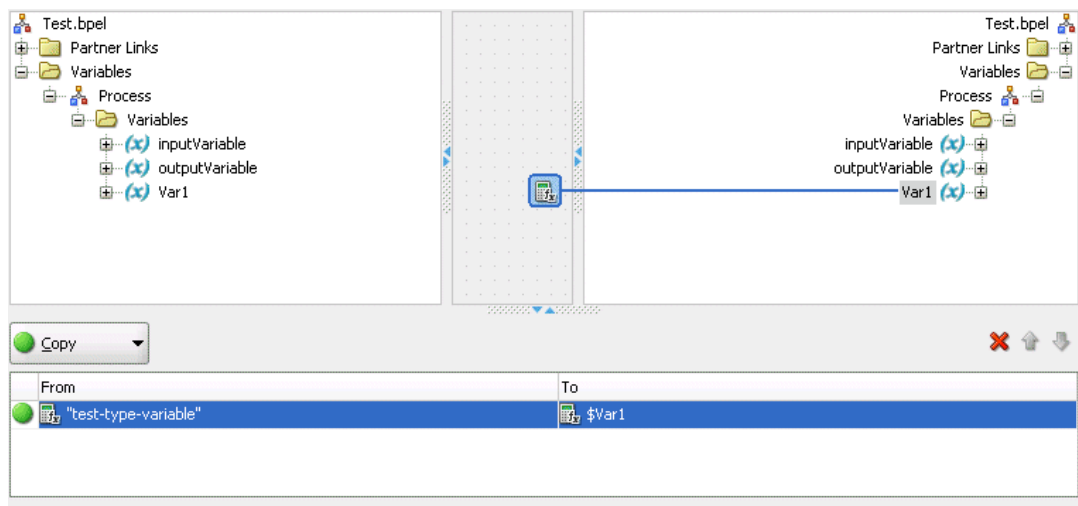
This section provides an overview of a simple BPEL process in which a reply activity uses the `toParts` elements to copy variable contents. The WSDL and BPEL files used in this example are shown later in [Example 6–74](#) and [Example 6–75](#) of [Section 6.17.2](#), "What Happens When You Map WSDL Message Parts."

How to map WSDL message parts in BPEL 2.0

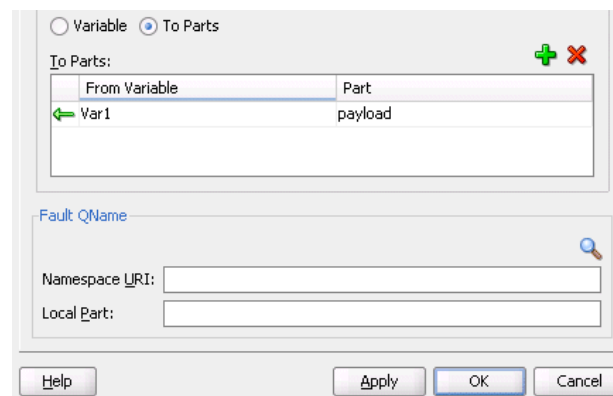
1. Note the receive activity in [Figure 6–11](#) includes a standard **inputVariable** variable from the client.

Figure 6–11 Receive Activity

2. Note the assign activity in [Figure 6–12](#) copies the **test-type-variable** contents to **Var1**.

Figure 6–12 Assign Activity

3. Note that the **To Parts** button at the bottom of the reply activity is enabled in [Figure 6–13](#), instead of the **Variable** button. You create information for this section by clicking the **Add** icon. The copy operation copies data from the variable indicated in the **From Variable** attribute, **Var1**, into the part of the anonymous, temporary WSDL variable referenced in the **Part** attribute.

Figure 6–13 To Parts Section Defined at Bottom of Reply Activity

6.17.2 What Happens When You Map WSDL Message Parts

[Example 6–74](#) shows a .bpel file for a synchronous request with `toPart` elements defined in a reply activity. This maps to the operation defined in the WSDL file shown in [Example 6–75](#). The copy operation copies data from the variable indicated in the `fromVariable` attribute into the part of the anonymous, temporary WSDL variable, `Var1`.

Example 6–74 BPEL File with ToParts Elements

```
<sequence name="main">
  <!-- Receive input from requestor. This maps to operation defined in WSDL -->
  <receive name="receiveInput" partnerLink="test_client"
    portType="client:Test" operation="process" variable="inputVariable"
    createInstance="yes"/>
  <!-- Generate reply to synchronous request -->
```

```

<assign name="Assign_1">
  <copy>
    <from>"test-type-variable"</from>
    <to>$Var1</to>
  </copy>
</assign>
<reply name="replyOutput" partnerLink="test_client" portType="client:Test"
  operation="process">
  <toParts>
    <toPart part="payload" fromVariable="Var1"/>
  </toParts>
</reply>
</sequence>

```

Example 6-75 WSDL File that Defines the Operation

```

<wsdl:types>
  <schema attributeFormDefault="unqualified" elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.com/RT_Validate_P_02_jws/ch10_
3toParts_1/Test"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="process">
      <complexType>
        <sequence>
          <element name="input" type="string"/>
        </sequence>
      </complexType>
    </element>
    <element name="processResponse">
      <complexType>
        <sequence>
          <element name="result" type="string"/>
        </sequence>
      </complexType>
    </element>
  </schema>
</wsdl:types>
<!-- ~~~~~
MESSAGE TYPE DEFINITION - Definition of the message types used as
part of the port type defintions
~~~~~ -->

<wsdl:message name="TestRequestMessage">
  <wsdl:part name="payload" element="client:process"/>
</wsdl:message>
<wsdl:message name="TestResponseMessage">
  <wsdl:part name="payload" type="xsd:string"/>
</wsdl:message>
<!-- ~~~~~
PORT TYPE DEFINITION - A port type groups a set of operations into
a logical service unit.
~~~~~ -->

<!-- portType implemented by the Test BPEL process -->
<wsdl:portType name="Test">
  <wsdl:operation name="process">
    <wsdl:input message="client:TestRequestMessage"/>
    <wsdl:output message="client:TestResponseMessage"/>
  </wsdl:operation>
</wsdl:portType>
<!-- ~~~~~
PARTNER LINK TYPE DEFINITION

```

```

~~~~~>
<plnk:partnerLinkType name="Test">
  <plnk:role name="TestProvider">
    <plnk:portType name="client:Test"/>
  </plnk:role>
</plnk:partnerLinkType>
</wsdl:definitions>

```

[Example 6-76](#) shows a .bpel file with `toPart` elements defined in invoke and reply activities. This maps to the operation defined in the WSDL file shown in [Example 6-77](#). The copy operation in the invoke activity copies data from the variable indicated in the `fromVariable` attribute into the part of the anonymous, temporary WSDL variable, `request`. The copy operation in the reply activity copies data from the variable indicated in the `fromVariable` attribute into the part of the anonymous, temporary WSDL variable, `output`.

Example 6-76 BPEL File with ToParts Elements

```

<sequence>
  <!-- receive input from requestor -->
  <receive name="receiveInput" partnerLink="client" portType="tns:Test"
    operation="process" variable="input" createInstance="yes"/>
  <assign>
    <copy>
      <from>$input.payload</from>
      <to>$request</to>
    </copy>
  </assign>
  <invoke name="invokeDummyService" partnerLink="DummyService"
    portType="tns:DummyPortType"
    operation="process" outputVariable="response">
    <toParts>
      <toPart part="payload" fromVariable="request" />
    </toParts>
  </invoke>
  <assign>
    <copy>
      <from>$response</from>
      <to>$output</to>
    </copy>
  </assign>
  <!-- respond output to requestor -->
  <reply name="replyOutput" partnerLink="client"
    portType="tns:Test" operation="process">
    <toParts>
      <toPart part="payload" fromVariable="output" />
    </toParts>
  </reply>
</sequence>

```

Example 6-77 WSDL File that Defines the Operation

```

<?xml version="1.0"?>
<definitions name="ch10.3toParts"
  targetNamespace="http://samples.otn.com/bpel2.0/ch10.3"
  xmlns:tns="http://samples.otn.com/bpel2.0/ch10.3"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
>

```

```
<types>
  <schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://samples.otn.com/bpel2.0/ch10.3"
xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="input" type="string"/>
    <element name="output" type="string"/>
  </schema>
</types>

<message name="TestRequestMessage">
  <part name="payload" element="tns:input"/>
</message>
<message name="TestResultMessage">
  <part name="payload" element="tns:output"/>
</message>
<portType name="Test">
  <operation name="process">
    <input message="tns:TestRequestMessage"/>
    <output message="tns:TestResultMessage"/>
  </operation>
</portType>

<plnk:partnerLinkType name="Test">
  <plnk:role name="TestProvider" portType="tns:Test"/>
</plnk:partnerLinkType>

</definitions>
```

[Example 6-78](#) shows a .bpel file with `fromParts` elements defined in pick and invoke activities. This maps to the operation defined in the WSDL file shown in [Example 6-79](#). The copy operation in the pick activity retrieves data from the variable indicated in the `toVariable` attribute into the part of the anonymous, temporary WSDL variable, `request`. The copy operation in the invoke activities retrieves data from the variable indicated in the `toVariable` attribute into the part of the anonymous, temporary WSDL variable, `response`.

Example 6-78 BPEL File with FromParts Elements

```
<sequence>
  <!-- receive input from requestor -->
  <pick createInstance="yes">
    <onMessage partnerLink="client" portType="tns:Test"
      operation="process">
      <fromParts>
        <fromPart part="payload" toVariable="request"/>
      </fromParts>
      <empty/>
    </onMessage>
  </pick>
  <invoke name="invokeDummyService" partnerLink="DummyService"
    portType="tns:DummyPortType"
    operation="process" inputVariable="request">
    <fromParts>
      <fromPart part="payload" toVariable="response"/>
    </fromParts>
  </invoke>
  <assign>
    <copy>
      <from>concat($response, " ", $response)</from>
      <to>$request</to>
    </copy>
  </assign>
</sequence>
```

```

        </copy>
    </assign>
    <invoke name="invokeDummyService" partnerLink="DummyService"
        portType="tns:DummyPortType"
        operation="process2" inputVariable="request">
        <fromParts>
            <fromPart part="payload" toVariable="response"/>
        </fromParts>
    </invoke>
    <assign>
        <copy>
            <from>$response</from>
            <to>$output.payload</to>
        </copy>
    </assign>
    <!-- respond output to requestor -->
    <reply name="replyOutput" partnerLink="client"
        portType="tns:Test" operation="process" variable="output"/>
</sequence>

```

Example 6-79 WSDL File that Defines the Operation

```

<?xml version="1.0"?>
<definitions name="BPEL20TestCh10.4"
    targetNamespace="http://samples.otn.com/bpel2.0/ch10.4"
    xmlns:tns="http://samples.otn.com/bpel2.0/ch10.4"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    >

    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified"
            targetNamespace="http://samples.otn.com/bpel2.0/ch10.4"
            xmlns="http://www.w3.org/2001/XMLSchema">
            <element name="input" type="string"/>
            <element name="output" type="string"/>
        </schema>
    </types>

    <message name="TestRequestMessage">
        <part name="payload" element="tns:input"/>
    </message>
    <message name="TestResultMessage">
        <part name="payload" element="tns:output"/>
    </message>
    <portType name="Test">
        <operation name="process">
            <input message="tns:TestRequestMessage"/>
            <output message="tns:TestResultMessage"/>
        </operation>
    </portType>

    <plnk:partnerLinkType name="Test">
        <plnk:role name="TestProvider" portType="tns:Test"/>
    </plnk:partnerLinkType>

</definitions>

```

6.18 Importing Process Definitions in BPEL 2.0

You can use the `import` element to specify the definitions on which your BPEL process is dependent. When you create a version 2.0 BPEL process, an `import` element is added to the `.bpel` file, as shown in [Example 6–80](#).

Example 6–80 Import Element

```
<process name="Loan Flow"
  . . .
  . . .
  <import namespace="http://xmlns.oracle.com/SOAApplication/SOAProject/LoanFlow"
    location="LoanFlow.wsdl" importType="http://schemas.xmlsoap.org/wsdl/" />
```

You can also use the `import` element to import a schema without a namespace, as shown in [Example 6–81](#).

Example 6–81 Schema Import Without Namespace

```
<process name="Loan Flow"
  . . .
  . . .
  <import location="xsd/NoNamespaceSchema.xsd"
    importType="http://www.w3.org/2001/XMLSchema" />
```

You can also use the `import` element to import a schema with a namespace, as shown in [Example 6–82](#).

Example 6–82 Schema Import With Namespace

```
<process name="Loan Flow"
  . . .
  . . .
  <import namespace="http://www.example.org" location="xsd/TestSchema.xsd"
    importType="http://www.w3.org/2001/XMLSchema" />
```

The `import` element is provided to declare a dependency on external XML schema or WSDL definitions. Any number of `import` elements can appear as children of the `process` element. Each `import` element can contain the following attributes.

- **namespace:** Identifies an absolute URI that specifies the imported definitions. This is an optional attribute. If a namespace is specified, then the imported definitions must be in that namespace. If a namespace is not specified, this indicates that external definitions are in use that are not namespace-qualified. The imported definitions must not contain a `targetNamespace` specification.
- **location:** Identifies a URI that specifies the location of a document containing important definitions. This is an optional attribute. This can be a relative URI. If no `location` attribute is specified, the process uses external definitions. However, there is no statement provided indicating where to locate these definitions.
- **importType:** Identifies the document type to import. This must be an absolute URI that specifies the encoding language used in the document. This is a required attribute.
 - If importing XML schema 1.0 documents, this attribute's value must be set to `"http://www.w3.org/2001/XMLSchema"`.
 - If importing WSDL 1.1 documents, the value must be set to `"http://schemas.xmlsoap.org/wsdl/"`. You can also specify other values for this attribute.

For more information, see section 5.4 of the *Web Services Business Process Execution Language Specification Version 2.0*.

6.19 Manipulating XML Data Sequences That Resemble Arrays

Data sequences are one of the most basic data models used in XML. However, manipulating them can be nontrivial. One of the most common data sequence patterns used in BPEL process service components are arrays. Based on the XML schema, the way you can identify a data sequence definition is by its attribute `maxOccurs` being set to a value greater than one or marked as unbounded. See the *XML Schema Specification* at <http://www.w3.org/TR> for more information.

The examples in this section illustrate several basic ways of manipulating data sequences in BPEL. However, there are other associated requirements, such as performing looping or dynamic referencing of endpoints. The following sections describe a particular requirement for data sequence manipulation.

6.19.1 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 6–83](#), `addresses[1]` selects the first element of the `addresses` data sequence:

Example 6–83 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 6–84](#) 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 6–84 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.

6.19.2 How to Use SOAP-Encoded Arrays

Oracle SOA Suite provides support for SOAP RPC-encoded arrays. This support enables Oracle BPEL Process Manager to operate as a client calling a SOAP web service (RPC-encoded) that uses a SOAP 1.1 array.

[Example 6–85](#) provides an example of a SOAP array payload named `myFavoriteNumbers`.

Example 6–85 SOAP Array Payload

```
<myFavoriteNumbers SOAP-ENC:arrayType="xsd:int2">
  <number>3</number>
  <number>4</number>
</myFavoriteNumbers>
```

In addition, ensure that the schema element attributes `attributeFormDefault` and `elementFormDefault` are set to `"unqualified"` in your schema. [Example 6–86](#) provides details:

Example 6–86 Schema Element Attributes

```
attributeFormDefault="unqualified" elementFormDefault="unqualified"
targetNamespace="java:services" xmlns:s0="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

The following features are not supported:

- A service published by BPEL that uses a SOAP array
- Partially-transmitted arrays
- Sparse arrays
- Multidimensional arrays

To use a SOAP-encoded array:

[Example 6–87](#) shows how to prepare SOAP arrays with the `bpelx:append` tag in a BPEL project.

1. Create a BPEL process in Oracle JDeveloper.
2. Prepare the payload for the invocation. Note that `bpelx:append` in [Example 6–87](#) is used to add items into the SOAP array.

Example 6–87 SOAP Array

```
<bpws:assign>
  <bpws:copy>
    <bpws:from variable="input" part="payload" query="/tns:value"/>
    <bpws:to variable="request" part="strArray"
      query="/strArray/JavaLangstring"/>
  </bpws:copy>
</bpws:assign>
<bpws:assign>
  <bpelx:append>
```

```

    <bpelx:from variable="request" part="strArray"
    query="/strArray/JavaLangstring1"/>
    <bpelx:to variable="request" part="strArray" query="/strArray"/>
  </bpelx:append>
</bpws:assign>

```

3. Import the following namespace in your WSDL file. Oracle JDeveloper does not understand the SOAP-ENC tag if the import statement is missing in the WSDL schema element.

```
<xs:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
```

6.19.2.1 SOAP-Encoded Arrays in BPEL 2.0

SOAP-encoded arrays are supported in BPEL projects that use version 2.0 of the BPEL specification. [Example 6–88](#) shows a sample assign activity with a SOAP-encoded array in a BPEL 2.0 project.

Example 6–88 SOAP-Encoded Array in an Assign Activity in BPEL 2.0

```

<assign name="Assign_1">
  <copy>
    <from>$inputVariable.payload</from>
    <to>$Invoke_1_echoArray_InputVariable.strArray/JavaLangstring[1]</to>
  </copy>
  <extensionAssignOperation>
    <bpelx:append>
      <bpelx:from variable="Invoke_1_echoArray_InputVariable"
        part="strArray">
        <bpelx:query>
          JavaLangstring[1]
        </bpelx:query>
      </bpelx:from>
      <bpelx:to variable="Invoke_1_echoArray_InputVariable"
        part="strArray">
      </bpelx:to>
    </bpelx:append>
  </extensionAssignOperation>
</assign>

```

[Example 6–89](#) shows a sample invoke activity with a SOAP-encoded array in a BPEL 2.0 project.

Example 6–89 SOAP-Encoded Array in an Invoke Activity in BPEL 2.0

```

<invoke name="Invoke1" partnerLink="FileOut"
  portType="ns3:Write_ptt" operation="Write"
  bpelx:invokeAsDetail="no">
  <toParts>
    <toPart part="body" fromVariable="ArrayVariable"/>
  </toParts>
</invoke>

```

6.19.3 How to Determine Sequence Size

If you must know the runtime size of a data sequence (that is, the number of nodes or data items in the sequence), you can get it by using the combination of the XPath built-in `count()` function and the BPEL built-in `getVariableData()` function.

The code in [Example 6–90](#) calculates the number of elements in the `item` sequence and assigns it to the integer variable `lineItemSize`.

Example 6–90 Sequence Size Determination

```
<assign>
  <copy>
    <from expression="count(bpws:getVariableData('outpoint', 'payload',
      '/p:invoice/p:lineItems/p:item'))"/>
    <to variable="lineItemSize"/>
  </copy>
</assign>
```

6.19.4 How to Dynamically Index by Applying a Trailing XPath to an Expression

Often a dynamic value is needed to index into a data sequence; that is, you must get the *n*th node out of a sequence, where the value of *n* is defined at runtime. This section covers the methods for dynamically indexing by applying a trailing XPath into expressions.

6.19.4.1 Applying a Trailing XPath to the Result of `getVariableData`

The dynamic indexing method shown in [Example 6–91](#) 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 6–91 Dynamic Indexing

```
<variable name="idx" type="xsd:integer"/>
...
<assign>
  <copy>
    <from expression="bpws:getVariableData('input', 'payload'
      )/p:line-item[bpws:getVariableData('idx')]/p:line-total" />
    <to variable="lineTotalVar" />
  </copy>
</assign>
```

Assume at runtime that the `idx` integer variable holds 2 as its value. The preceding expression within the `from` is equivalent to that shown in [Example 6–92](#).

Example 6–92 Equivalent Format

```
<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.

[Example 6–93](#) provides details.

Example 6–93 Root Element Name Specification

```
bpws:getVariableData('input', 'payload',
```

```
'/p:invoice/p:line-item[2]/p:line-total')
```

If the XPath is used trailing behind the `bpws:getVariableData()` function, the root element name does not need to be specified in the XPath.

For example:

```
bpws:getVariableData('input', 'payload')/p:line-item[2]/p:line-total
```

This is because the node returned by the `getVariableData()` function is the root element. Specifying the root element name again in the XPath is redundant and is incorrect according to standard XPath semantics.

6.19.4.2 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 6-94](#) provides an example.

Example 6-94 *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 the `output` variable is used as the parent element.

6.19.4.3 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 6-95](#) under `assign` demonstrate how to merge data sequences:

Example 6-95 *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>
```

6.19.4.4 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 6–96](#) shows an `append` statement initializing a purchase order (PO) document with 10 empty `<lineItem>` elements under `po`:

Example 6–96 *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 [Example 6–96](#) can be replaced with an embedded XQuery expression, as shown in [Example 6–97](#).

Example 6–97 *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.

6.19.5 What You May Need to Know About Using the Array Identifier

For processing in Native Format Builder array identifier environments, information is required about the parent node of a node. Because the `reportSAXEvents` API is used, this information is typically not available for outbound message scenarios. Setting `nxsd:useArrayIdentifiers` to `true` in the native schema enables DOM-parsing to be used for outbound message scenarios. Use this setting cautiously, as it can lead to slower performance for very large payloads. [Example 6–98](#) provides details.

Example 6–98 Array Identifier

```
<?xml version="1.0" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://xmlns.oracle.com/pcbpel/demoSchema/csv"
            xmlns:tns="http://xmlns.oracle.com/pcbpel/demoSchema/csv"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified" nxsd:encoding="US-ASCII"
nxsd:stream="chars" nxsd:version="NXSD" nxsd:useArrayIdentifiers="true">
  <xsd:element name="Root-Element">
    ....
  </xsd:element>
</xsd:schema>
```

6.20 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 a string type. With Java, you use DOM functions to convert the string to a structured XML object type. You can use the BPEL XPath function `parseEscapedXML` to do the same thing.

For information about `parseEscapedXML`, see [Section B.2.50, "parseEscapedXML."](#)

6.20.1 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 6–99](#) provides an example:

Example 6–99 String to XML Element Conversion

```
<!-- execute the XPath extension function
parseEscapedXML('&lt;item&gt;') and assign to a variable
-->
<assign>
  <copy>
    <from expression="oratext: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>>2000
        </lineTotal>
        </item>' ) "/>
    <to variable="escapedLineItem"/>
</copy>
</assign>
```

6.21 Understanding Document-Style and RPC-Style WSDL Differences

The examples shown up to this point have been for document-style WSDL files in which a message is defined with an XML schema element, as shown in [Example 6–100](#):

Example 6–100 XML Schema element Definition

```
<message name="LoanFlowRequestMessage">
  <part name="payload" element="s1:loanApplication"/>
</message>
```

This is in contrast to RPC-style WSDL files, in which the message is defined with an XML schema type, as shown in [Example 6–101](#):

Example 6–101 RPC-Style type Definition

```
<message name="LoanFlowRequestMessage">
  <part name="payload" type="s1:LoanApplicationType"/>
</message>
```

6.21.1 How To Use RPC-Style Files

This impacts the material in this chapter because there is a difference in how XPath queries are constructed for the two WSDL message styles. For an RPC-style message, the top-level element (and therefore the first node in an XPath query string) is the part name (payload in [Example 6–101](#)). In document-style, the top-level node is the element name (for example, loanApplication).

[Example 6–102](#) and [Example 6–103](#) show what an XPath query string looks like if an application named LoanServices were in RPC style.

Example 6–102 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 6–103 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>

```

6.22 Manipulating SOAP Headers in BPEL

BPEL's communication activities (invoke, receive, reply, and onMessage) receive and send messages through specified message variables. These default activities permit one variable to operate in each direction. For example, the invoke activity has `inputVariable` and `outputVariable` attributes. You can specify one variable for each of the two attributes. This is enough if the particular operation involved uses only one payload message in each direction.

However, WSDL supports multiple messages in an operation. In the case of SOAP, multiple messages can be sent along the main payload message as SOAP headers. However, BPEL's default communication activities cannot accommodate the additional header messages.

Oracle BPEL Process Manager solves this problem by extending the default BPEL communication activities with the `bpelx:headerVariable` extension. The extension syntax is as shown in [Example 6-104](#):

Example 6-104 *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 ..." .../>

```

6.22.1 How to Receive SOAP Headers in BPEL

This section provides an example of how to create BPEL and WSDL files to receive SOAP headers.

To receive SOAP headers in BPEL:

1. Create a WSDL file that declares header messages and the SOAP binding that binds them to the SOAP request. [Example 6-105](#) provides an example.

Example 6-105 *WSDL File Contents*

```

<!-- custom header -->
<message name="CustomHeaderMessage">
  <part name="header1" element="tns:header1"/>
  <part name="header2" element="tns:header2"/>
</message>

<binding name="HeaderServiceBinding" type="tns:HeaderService">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="initiate">
    <soap:operation style="document" soapAction="initiate"/>
    <input>
      <soap:header message="tns:CustomHeaderMessage"

```

```
        part="header1" use="literal"/>
      <soap:header message="tns:CustomHeaderMessage"
        part="header2" use="literal"/>
    <soap:body use="literal"/>
  </input>
</operation>
</binding>
```

2. Create a BPEL source file that declares the header message variables and uses `bpelx:headerVariable` to receive the headers, as shown in [Example 6–106](#).

Example 6–106 *bpelx:headerVariable* Use

```
<variables>  <variable name="input"
              messageType="tns:HeaderServiceRequestMessage"/>
  <variable name="event"
            messageType="tns:HeaderServiceEventMessage"/>
  <variable name="output"
            messageType="tns:HeaderServiceResultMessage"/>
  <variable name="customHeader"
            messageType="tns:CustomHeaderMessage"/>
</variables>

<sequence>
  <!-- receive input from requester -->
  <receive name="receiveInput" partnerLink="client"
    portType="tns:HeaderService" operation="initiate"
    variable="input"
    bpelx:headerVariable="customHeader"
    createInstance="yes"/>
```

6.22.2 How to Send SOAP Headers in BPEL

This section provides an example of how to send SOAP headers.

To send SOAP headers in BPEL:

1. Define an SCA reference in the `composite.xml` to refer to the `HeaderService`.
2. Define the custom header variable, manipulate it, and send it using `bpelx:inputHeaderVariable`, as shown in [Example 6–107](#).

Example 6–107 *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"/>
```

6.23 Declaring Extension Namespaces in BPEL 2.0

You can extend a version 2.0 BPEL process to add custom extension namespace declarations. With the `mustUnderstand` attribute, you can indicate whether the custom namespaces carry semantics that must be understood by the BPEL process.

If a BPEL process does not support one or more of the extensions with `mustUnderstand` set to `yes`, the process definition is rejected.

Extensions are defined in the `extensions` element. [Example 6–108](#) provides details.

Example 6–108 Extension Namespace Declaration Syntax

```
<process ...>
  ...
  <extensions>?
    <extension namespace="myURI" mustUnderstand="yes|no" />+
  </extensions>
  ...
</process>
```

The contents of an `extension` element must be a single element qualified with a namespace different from the standard BPEL namespace.

For more information about extension declarations, see the *Web Services Business Process Execution Language Version 2.0 Specification* located at the following URL:

<http://www.oasis-open.org>

6.23.1 How to Declare Extension Namespaces

To declare extension namespaces:

1. In a BPEL 2.0 process, click the **Extensions** icon above Oracle BPEL Designer.
The Extensions dialog is displayed.
2. Select the **Extensions** folder, then click the **Add** icon.
The Extension dialog is displayed.
3. In the **Namespace** field, enter the extension namespace to declare. This namespace must be different from the standard BPEL namespace.
4. If you want the extensions to be recognized by the BPEL process, select the **Must Understand** checkbox.
5. Click **OK**.
6. Click **Close**.

6.23.2 What Happens When You Create an Extension

After you complete your design, the `.bpe1` process looks as shown in [Example 6–109](#).

Example 6–109 Extension with Custom Namespace

```
<extensions>
  <extension namespace="http://xmlns.mycompany.com/myNamespace"
    mustUnderstand="yes" />
</extensions>
```

Invoking a Synchronous Web Service from a BPEL Process

This chapter describes how to invoke a synchronous web service from a BPEL process. This chapter demonstrates how to set up the components necessary to perform a synchronous invocation. This chapter also examines how these components are coded.

This chapter includes the following sections:

- [Section 7.1, "Introduction to Invoking a Synchronous Web Service"](#)
- [Section 7.2, "Invoking a Synchronous Web Service"](#)
- [Section 7.3, "Specifying Timeout Values"](#)
- [Section 7.4, "Calling a One-Way Mediator with a Synchronous BPEL Process"](#)

For a simple Hello World sample (bpe1-101-HelloWorld) that takes an input string, adds a prefix of "Hello " to the string, and returns it, visit the following URL:

<https://soasamples.samplecode.oracle.com/>

7.1 Introduction to Invoking a Synchronous Web Service

Synchronous web services provide an immediate response to an invocation. BPEL can connect to synchronous web services through a partner link, send data, and then receive the reply in the same synchronous invocation.

A synchronous invocation requires the following components:

- Partner link

Defines the location and the role of the web services with which the BPEL process service component connects to perform tasks, and the variables used to carry information between the web service and the BPEL process service component. A partner link is required for each web service that the BPEL process service component calls. You can create partner links in several ways, including the following:

- In the SOA Composite Editor, when you drag a **Web Service** from the **Service Adapters** section of the Component Palette into the **Exposed Services** or **External References** swimlane. For more information, see [Section 2.3, "Adding Service Binding Components"](#) or [Section 2.4, "Adding Reference Binding Components."](#)
- In the Oracle BPEL Designer, when you drag a **Partner Link** from the **BPEL Constructs** section of the Component Palette into the **Partner Links** swimlane. This method is described in this chapter.

- Invoke activity

Opens a port in the BPEL process service component to send and receive data. For example, this port is used to retrieve information verifying that a customer has acceptable credit using a credit card authorization service. For synchronous callbacks, only one port is needed for both the send and receive functions.

7.2 Invoking a Synchronous Web Service

This section examines a synchronous invocation operation using the `OrderProcessor.bpel` file in the WebLogic Fusion Order Demo application as an example.

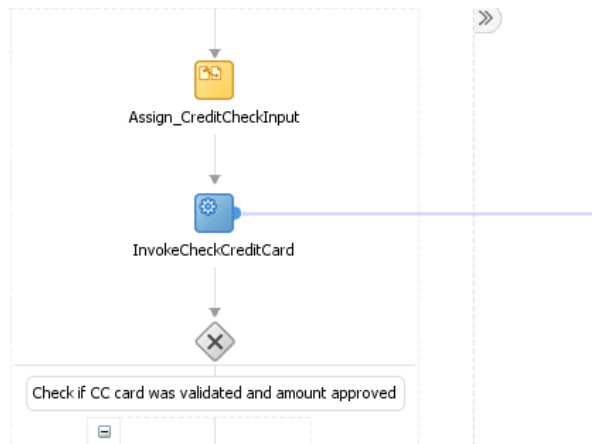
7.2.1 How to Invoke a Synchronous Web Service

To invoke a synchronous web service:

1. In the Component Palette in Oracle BPEL Designer, expand **BPEL Constructs**.
2. Drag the necessary partner link, invoke activity, scope activity, and assign activity into the designer.
3. Edit their dialogs.

[Figure 7-1](#) shows the diagram for the **Scope_AuthorizeCreditCard** scope activity of the **OrderProcessor.bpel** file in the Fusion Order Demo, which defines a simple set of actions.

Figure 7-1 *Diagram of OrderProcessor.bpel*



The following actions take place:

1. The **Assign_CreditCheckInput** **assign** activity packages the data from the client. The assign activity provides a method for copying the contents of one variable to another. In this case, it takes the credit card type, credit card number, and purchase amount and assigns them to the input variable for the **CreditAuthorizationService** service.
2. The **InvokeCheckCreditCard** activity calls the **CreditCardAuthorization** service. [Figure 7-2](#) shows the **CreditCardAuthorizationService** web service, which is defined as a partner link.

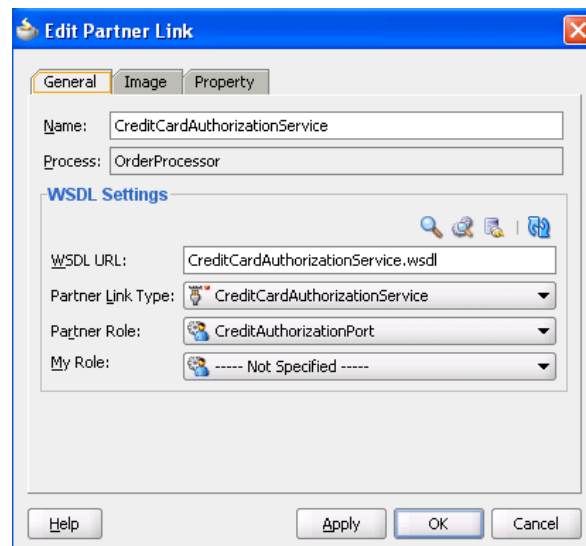
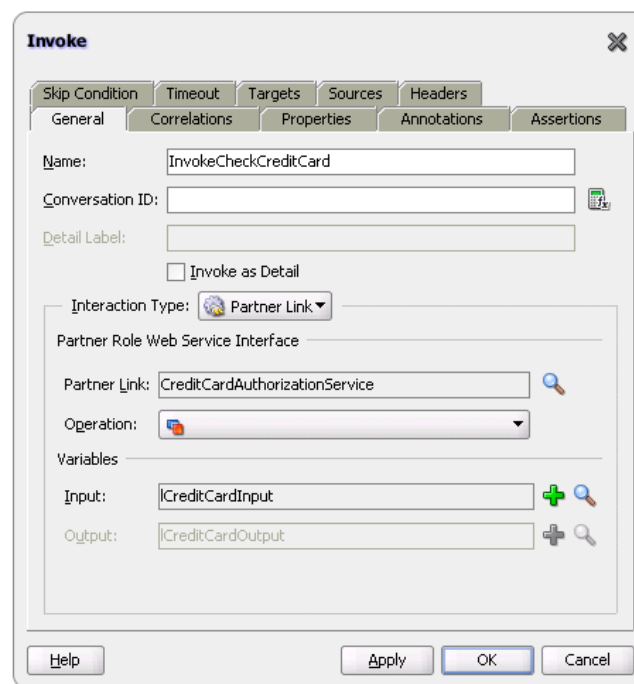
Figure 7-2 CreditCardAuthorizationService Partner Link

Figure 7-3 shows the **InvokeCheckCreditCard** invoke activity.

Figure 7-3 InvokeCheckCreditCard Invoke Activity

3. The **Switch_EvaluateCCResult** switch activity in Figure 7-1 checks the results of the credit card validation. For information about switch activities, see [Section 10.2.1, "Defining Conditional Branching with the Switch Activity in BPEL 1.1."](#)

Note: The switch activity is replaced by the if activity in BPEL 2.0.

7.2.2 What Happens When You Invoke a Synchronous Web Service

When you create a partner link and invoke activity, the necessary BPEL code for invoking a synchronous web service is added to the appropriate BPEL and Web Services Description Language (WSDL) files.

7.2.2.1 Partner Link in the BPEL Code

In the `OrderProcessor.bpel` code, the partner link defines the link name and type, and the role of the BPEL process service component in interacting with the partner service.

From the BPEL source code, the `CreditCardAuthorizationService` partner link definition is shown in [Example 7-1](#):

Example 7-1 Partner Link Definition

```
<partnerLink name="CreditCardAuthorizationService"
  partnerRole="CreditAuthorizationPort"
  partnerLinkType="ns2:CreditCardAuthorizationService"/>
```

Variable definitions that are accessible locally in the `Scope_AuthorizeCreditCard` scope are shown in [Example 7-2](#). The types for these variables are defined in the WSDL for the process itself.

Example 7-2 Variable Definition

```
<variable name="lCreditCardInput"
  messageType="ns2:CreditAuthorizationRequestMessage"/>
<variable name="lCreditCardOutput"
  messageType="ns2:CreditAuthorizationResponseMessage"/>
```

The WSDL file defines the interface to your BPEL process service component: the messages that it accepts and returns, the operations that are supported, and other parameters.

7.2.2.2 Partner Link Type and Port Type in the BPEL Code

The web service's `CreditCardAuthorizationService.wsdl` file contains two sections that enable the web service to work with BPEL process service components:

- `partnerLinkType`:
 - Defines the following characteristics of the conversion between a BPEL process service component and the credit card authorization web service:
 - The role (operation) played by each
 - The `portType` provided by each for receiving messages within the conversation
- `portType`:
 - A collection of related operations implemented by a participant in a conversation. A port type defines which information is passed back and forth, the form of that information, and so on. A synchronous invocation requires only one port type that both initiates the synchronous process and calls back the client with the response. An asynchronous callback (one in which the reply is not immediate) requires two port types, one to send the request, and another to receive the reply when it arrives.

In this example, the portType `CreditAuthorizationPort` receives the credit card type, credit card number, and purchase amount, and returns the status results.

[Example 7-3](#) provides an example of `partnerLinkType` and `portType`.

Example 7-3 *partnerLinkType and portType Definitions*

```
<plnk:partnerLinkType name="CreditCardAuthorizationService">
  <plnk:role name="CreditAuthorizationPort">
    <plnk:portType name="tns:CreditAuthorizationPort"/>
  </plnk:role>
</plnk:partnerLinkType>
```

7.2.2.3 Invoke Activity for Performing a Request

The invoke activity includes the `lCreditCardInput` local input variable. The credit card authorization web service uses the `lCreditCardInput` input variable. This variable contains the customer's credit card type, credit card number, and purchase amount. The `lCreditCardOutput` variable returns status results from the `CreditAuthorizationService` service. [Example 7-4](#) provides an example.

Example 7-4 *Invoke Activity*

```
<invoke name="InvokeCheckCreditCard"
  inputVariable="lCreditCardInput"
  outputVariable="lCreditCardOutput"
  partnerLink="CreditCardAuthorizationService"
  portType="ns2:CreditAuthorizationPort"
  operation="AuthorizeCredit"/>
```

7.2.2.4 Synchronous Invocation in BPEL Code

The BPEL code shown in [Example 7-5](#) performs the synchronous invocation:

Example 7-5 *Synchronous Invocation*

```
<assign name="Assign_CreditCheckInput">
  <copy>
    <from variable="gOrderInfoVariable"
      query="/ns4:orderInfoVOSDO/ns4:OrderTotal"/>
    <to variable="lCreditCardInput" part="Authorization"
      query="/ns8:AuthInformation/ns8:PurchaseAmount"/>
  </copy>
  <copy>
    <from variable="gOrderInfoVariable"
      query="/ns4:orderInfoVOSDO/ns4:CardTypeCode"/>
    <to variable="lCreditCardInput" part="Authorization"
      query="/ns8:AuthInformation/ns8:CCType"/>
  </copy>
  <copy>
    <from variable="gOrderInfoVariable"
      query="/ns4:orderInfoVOSDO/ns4:AccountNumber"/>
    <to variable="lCreditCardInput" part="Authorization"
      query="/ns8:AuthInformation/ns8:CCNumber"/>
  </copy>
</assign>
<invoke name="InvokeCheckCreditCard"
  inputVariable="lCreditCardInput"
  outputVariable="lCreditCardOutput"
  partnerLink="CreditCardAuthorizationService"
```

```
portType="ns2:CreditAuthorizationPort"
operation="AuthorizeCredit"/>
```

7.3 Specifying Timeout Values

You can specify timeout values with the property **SyncMaxWaitTime** in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control. This property defines the maximum time a request and response operation takes before timing out. If the BPEL process service component does not receive a reply within the specified time, then the activity fails.

7.3.1 How To Specify Timeout Values

To specify timeout values:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control.
2. From the **SOA Infrastructure** menu, select **SOA Administration > BPEL Properties**.
3. At the bottom of the BPEL Service Engine Properties page, click **More BPEL Configuration Properties**.
4. Click **SyncMaxWaitTime**.
5. In the **Value** field, specify a value in seconds.
6. Click **Apply**.
7. Click **Return**.

7.3.2 What You May Need to Know About SyncMaxWaitTime and Synchronous Requests Not Timing Out

The **SyncMaxWaitTime** property applies to durable processes that are called in an asynchronous manner.

Assume you have a BPEL process with the definition shown in [Example 7-6](#). The process is not durable because there are no breakpoint activities.

Example 7-6 Process with No Breakpoint Activities

```
<receive name="receiveInput" partnerLink="client" variable="input"
createInstance="yes" />
<assign>
...
</assign>
<reply name="replyOutput" partnerLink="client" variable="output" />
```

If a Java client or another BPEL process calls this process, the assign activity is performed and the reply activity sets the output message into a HashMap for the client (actually the delivery service) to retrieve. Since the reply is the last activity, the thread returns to the client side and tries to pick up the reply message. Since the reply message was previously inserted, the client does not wait and returns with the reply.

Assume you have a BPEL process with a breakpoint activity, as shown in [Example 7-7](#).

Example 7-7 Process with Breakpoint Activities

```
<receive name="receiveInput" partnerLink="client" variable="input"
```

```

createInstance="yes" />
<assign>
...
</assign>
<wait for="'PT10S'" />
<reply name="replyOutput" partnerLink="client" variable="output" />

```

While it is not recommended to have asynchronous activities inside a synchronous process, BPEL does not prevent this type of design.

When the client (or another BPEL process) calls the process, the wait (breakpoint) activity is executed. However, since the wait is processed after some time by an asynchronous thread in the background, the executing thread returns to the client side. The client (actually the delivery service) tries to pick up the reply message, but it is not there since the reply activity in the process has not yet executed. Therefore, the client thread waits for the **SyncMaxWaitTime** seconds value. If this time is exceeded, then the client thread returns to the caller with a timeout exception.

If the wait is less than the **SyncMaxWaitTime** value, the asynchronous background thread then resumes at the wait and executes the reply. The reply is placed in the HashMap and the waiter (the client thread) is notified. The client thread picks up the reply message and returns.

Therefore, **SyncMaxWaitTime** only applies to synchronous process invocations when the process has a breakpoint in the middle. If there is no breakpoint, the entire process is executed by the client thread and returns the reply message.

7.4 Calling a One-Way Mediator with a Synchronous BPEL Process

You can expose a synchronous interface in the front end while using an asynchronous callback in the back end to simulate a synchronous reply. This is the default behavior in BPEL processes with the automatic setting of the `configuration.transaction` property to `requiresNew` in the `composite.xml` file. [Example 7-8](#) provides details.

Example 7-8 *configuration.transaction Property*

```

<component name="BPELProcess1">
@ <implementation.bpel src="BPELProcess1.bpel"/>
@ <property name="configuration.transaction" type="xs:string"
@ many="false">requiresNew</property>
@ </component>

```

`RequiresNew` is the recommended value. If you want to participate in the client's transaction, you must set the `configuration.transaction` property to `Required`.

Invoking an Asynchronous Web Service from a BPEL Process

This chapter describes how to call an asynchronous web service. Asynchronous messaging styles are useful for environments in which a service, such as a loan processor, can take a long time to process a client request. Asynchronous services also provide a more reliable fault-tolerant and scalable architecture than synchronous services.

This chapter includes the following sections:

- [Section 8.1, "Introduction to Invoking an Asynchronous Web Service"](#)
- [Section 8.2, "Invoking an Asynchronous Web Service"](#)
- [Section 8.3, "Using a Dynamic Partner Link at Runtime"](#)
- [Section 8.4, "Using WS-Addressing in an Asynchronous Service"](#)
- [Section 8.5, "Using Correlation Sets in an Asynchronous Service"](#)

8.1 Introduction to Invoking an Asynchronous Web Service

This section introduces asynchronous web service invocation with a company called United Loan. United Loan publishes an asynchronous web service that processes a client's loan application request and then returns a loan offer. This use case discusses how to integrate a BPEL process service component with this asynchronous loan application approver web service.

This use case illustrates the key design concepts for requesting information from an asynchronous service, and then receiving the response. The asynchronous United Loan service in this example is another BPEL process service component. However, the same BPEL call can interact with any properly designed web service. The target web service WSDL file contains the information necessary to request and receive the necessary information.

For the asynchronous web service, the following actions take place (in order of priority):

1. An assign activity prepares the loan application.
2. An invoke activity initiates the loan request. The contents of this request are put into a request variable. This request variable is sent to the asynchronous loan processor web service.

When the loan request is initiated, a correlation ID unique to the client and partner link initiating the request is also sent to the loan processor web service. The

correlation ID ensures that the correct loan offer response is returned to the corresponding loan application requester.

3. The loan processor web service then sends the correct response to the receive activity, which has been tracked by the correlation ID.
4. An assign activity reads the loan application offer.

The remaining sections in this chapter provide specific details about the asynchronous functionality.

8.2 Invoking an Asynchronous Web Service

This section provides an overview of the tasks for adding asynchronous functionality to a BPEL process service component.

8.2.1 How to Invoke an Asynchronous Web Service

You perform the following steps to asynchronously invoke a web service:

- Add a partner link
- Add an invoke activity
- Add a receive activity
- Create assign activities

8.2.1.1 Adding a Partner Link for an Asynchronous Service

These instructions describe how to create a partner link in a BPEL process (for this example, named `LoanService`) for the loan application approver web service.

To add a partner link for an asynchronous service:

1. In the SOA Composite Editor, drag a BPEL process from the **Service Components** section of the Component Palette into the designer.

The Create BPEL Process dialog appears.

2. Follow the instructions in the dialog to create a BPEL process service component.
3. Click **OK** when complete.
4. In the SOA composite application in the SOA Composite Editor, double-click the BPEL process service component (for this example, the component is named **LoanBroker**).

The Oracle BPEL Designer appears.

5. In the Component Palette, expand **BPEL Constructs**.
6. Drag a **Partner Link** icon into the right **Partner Links** swimlane.

The Create Partner Link dialog appears.

7. Enter the following details to create a partner link and select the loan application approver web service:

- **Name**

Enter a name for the partner link (for this example, `LoanService` is entered).

- **Process**

Displays the BPEL process service component name (for this example, `LoanBroker` appears).

- **WSDL URL**

Enter the name of the Web Services Description Language (WSDL) file to use. Click the **SOA Resource Lookup** icon above this field to locate the correct WSDL.

- **Partner Link Type**

Refers to the external service with which the BPEL process service component is to interface. Select from the list (for this example, `LoanService` is selected).

- **Partner Role**

Refers to the role of the external source, for example, provider. Select from the list (for this example, `LoanServiceProvider` is selected).

- **My Role**

Refers to the role of the BPEL process service component in this interaction. Select from the list (for this example, `LoanServiceRequester` is selected).

8. Click **OK**.

A new partner link for the loan application approver web service (United Loan) appears in the swimlane of the designer.

8.2.1.2 Adding an Invoke Activity

Follow these instructions to create an invoke activity and a global input variable named `request`. This activity initiates the asynchronous BPEL process service component activity with the loan application approver web service (United Loan). The loan application approver web service uses the `request` input variable to receive the loan request from the client.

To add an invoke activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag an **Invoke** activity to beneath the **Receive** activity.
3. Go to the Structure window. Note that while this example describes variable creation from the Structure window, you can also create variables by clicking the **Add** icons to the right of the **Input** and **Output** fields of the Invoke dialog.
4. Right-click **Variables** and select **Expand All Child Nodes**.
5. In the second **Variables** folder in the tree, right-click and select **Create Variable**. The Create Variable dialog appears.
6. Enter the variable name and select **Message Type** from the options provided:
 - **Type**
This option lets you select an XML schema simple type (for example, string, boolean, and so on).
 - **Message Type**
This option enables you to select a WSDL message file definition of a partner link or of the project WSDL file of the current BPEL process service component (for example, a response message or a request message). You can specify

variables associated with message types as input or output variables for invoke, receive, or reply activities.

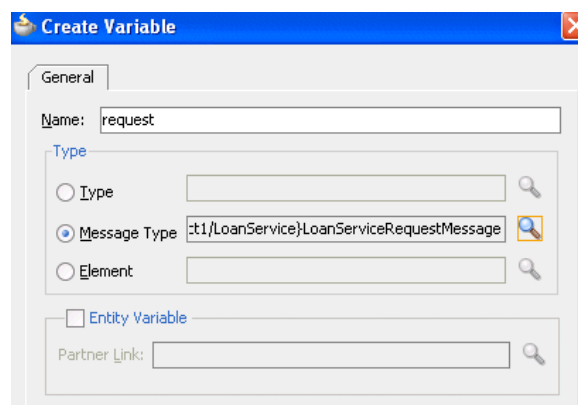
To display the message type, select the **Message Type** option, and then select its **Browse** icon to display the Type Chooser dialog. From here, expand the Message Types tree to make your selection. For this example, **Message Types > Partner Links > Loan Service > LoanService.wsdl > Message Types > LoanServiceRequestMessage** is selected.

■ Element

This option lets you select an XML schema element of the project schema file or project WSDL file of the current BPEL process service component, or of a partner link.

Figure 8–1 shows the Create Variable dialog.

Figure 8–1 Create Variable Dialog



7. Click **OK**.
8. Double-click the **invoke** activity to display the Invoke dialog.
9. In the Invoke dialog, select the partner link from the **Partner Link** list (for this example, **LoanService** is selected) and **initiate** from the **Operation** list.
10. To the right of the **Input** field, click the second icon and select the input variable you created in Step 6.

The Variable Chooser dialog appears, where you can select the variable.

There is no output variable specified because the output variable is returned in the receive operation. The **invoke** activity is created.

For more information about the invoke activity, see [Section 8.2.2.5, "Invoke and Receive Activities."](#)

11. Click **OK**.

8.2.1.3 Adding a Receive Activity

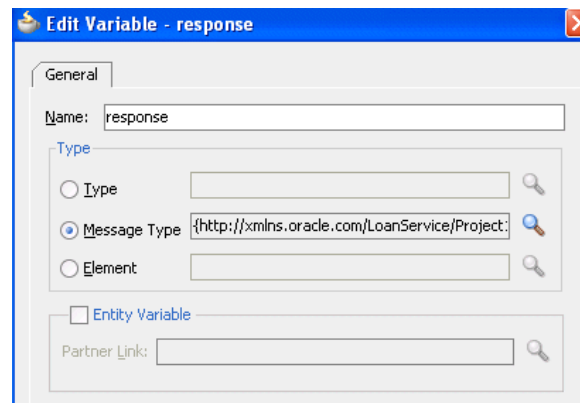
Follow these steps to create a receive activity and a global output variable named **response**. This activity waits for the loan application approver web service's callback operation. The loan application approver web service uses this output variable to send the loan offer result to the client.

To add a receive activity:

1. From the Component Palette, drag a **Receive** activity to the location right after the **Invoke** activity you created in [Section 8.2.1.2, "Adding an Invoke Activity."](#)
2. Create a variable to hold the receive information by invoking the Create Variable dialog, as you did in Step 3 through Step 7 of [Section 8.2.1.2, "Adding an Invoke Activity."](#)

Figure 8–2 shows the Create Variable dialog in BPEL 1.1.

Figure 8–2 Create Variable Dialog



Not : In BPEL projects that support version 2.0 of the BPEL specification, the Create Variable dialog includes an **Initialize** tab that enables you to initialize the variable type inline (for example, as a variable, expression, literal, partner link, or property). For more information, see [Section 6.5.2, "Initializing Variables with an Inline from-spec in BPEL 2.0."](#)

3. Double-click the **receive** activity and change its name to `receive_invoke`.
4. From the **Partner Link** list, select the partner link (for this example, **LoanService** is selected).
5. From the **Operation** list, select **onResult**. Do not select the **Create Instance** checkbox.
6. Select the variable you created in Step 3 through Step 7 of [Section 8.2.1.2, "Adding an Invoke Activity."](#)
7. Click **OK**.

The **receive** activity and the output variable are created. Because the initial **receive** activity in the BPEL file (for this example, **LoanBroker.bpel**) created the initial BPEL process service component instance, a second instance does not need to be created.

8.2.1.4 Performing Additional Activities

In addition to the asynchronous-specific tasks, you must perform the following tasks.

- Create an initial assign activity for data manipulation in front of the invoke activity that copies the client's **input** variable loan application request document

payload into the loan application approver web service's **request** variable payload.

- Create a second assign activity for data manipulation after the receive activity that copies the loan application approver web service's **response** variable loan application results payload into the **output** variable for the client to receive.

8.2.2 What Happens When You Invoke an Asynchronous Web Service

This section describes what happens when you invoke an asynchronous web service.

8.2.2.1 portType Section of the WSDL File

The `portType` section of the WSDL file (in this example, for `LoanService`) defines the ports to be used for the asynchronous service.

Asynchronous services have two port types. Each port type performs a one-way operation. In this example, one port type responds to the asynchronous process and the other calls back the client with the asynchronous response. In the example shown in [Example 8-1](#), the `portType` `LoanServiceCallback` receives the client's loan application request and the `portType` `LoanService` asynchronously calls back the client with the loan offer response.

Example 8-1 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>
```

8.2.2.2 partnerLinkType Section of the WSDL File

The `partnerLinkType` section of the WSDL file (in this example, for `LoanService`) defines the following characteristics of the BPEL process service component:

- The role (operation) played
- The `portType` provided for receiving messages within the conversation

Partner link types in asynchronous services have two roles: one for the web service provider and one for the client requester.

In the conversation shown in [Example 8-2](#), 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 8-2 partnerLinkType Definition

```
<plnk:partnerLinkType name="LoanService">
  <plnk:role name="LoanServiceProvider">
```

```

        <plnk:portType name="client:LoanService" />
    </plnk:role>
    <plnk:role name="LoanServiceRequester">
        <plnk:portType name="client:LoanServiceCallback" />
    </plnk:role>
</plnk:partnerLinkType>

```

Two port types are combined into this single asynchronous BPEL process service component: `portType="services:LoanService"` of the `invoke` activity and `portType="services:LoanServiceCallback"` of the `receive` activity. Port types are essentially a collection of operations to be performed. For this BPEL process service component, there are two operations to perform: `initiate` in the `invoke` activity and `onResult` in the `receive` activity.

8.2.2.3 Partner Links Section in the BPEL File

To call the service from BPEL, you use the BPEL file to define how the process interfaces with the web service. View the `partnerLinks` section. The services with which a process interacts are designed as partner links. Each partner link is characterized by a `partnerLinkType`.

Each partner link is named. This name is used for all service interactions through that partner link. This is critical in correlating responses to different partner links for simultaneous requests of the same type.

Asynchronous processes use a second partner link for the callback to the client. In this example, the second partner link, `LoanService`, is used by the loan application approver web service. [Example 8-3](#) provides an example.

Example 8-3 *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.

8.2.2.4 Composite Application File

In the `composite.xml` file, the loan application approver web service appears, as shown in [Example 8-4](#).

Example 8-4 *Loan Application Approver Web Service*

```

<component name="LoanBroker">
    <implementation.bpel process="LoanBroker.bpel" />
</component>

```

For more information, see [Section 8.2.1.1, "Adding a Partner Link for an Asynchronous Service"](#) for instructions on creating a partner link.

8.2.2.5 Invoke and Receive Activities

View the `variables` and `sequence` sections. Two areas of particular interest concern the `invoke` and `receive` activities:

- An `invoke` activity invokes a synchronous web service (as discussed in [Chapter 7, "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 8-5](#) provides an example.

Example 8-5 Invoke and Receive Activities

```
<variables>

  <variable name="request"
    messageType="services:LoanServiceRequestMessage" />
  <variable name="response"
    messageType="services:LoanServiceResultMessage" />
</variables>

<sequence>

  <!-- initialize the input of LoanService -->
  <assign>
    <!-- initiate the remote process -->
    <invoke name="invoke" partnerLink="LoanService"
      portType="services:LoanService"
      operation="initiate" inputVariable="request" />

    <!-- receive the result of the remote process -->
    <receive name="receive_invoke" partnerLink="LoanService"
      portType="services:LoanServiceCallback"
      operation="onResult" variable="response" />
```

When an asynchronous service is initiated with the `invoke` activity, a correlation ID unique to the client request is also sent, using Web Services Addressing (WS-Addressing) (described in [Section 8.4, "Using WS-Addressing in an Asynchronous Service"](#)). Because multiple processes may be waiting for service callbacks, the server must know which BPEL process service component instance is waiting for a callback message from the loan application approver web service. The correlation ID enables the server to correlate the response with the appropriate requesting instance.

8.2.2.6 createInstance Attribute for Starting a New Instance

You may notice a `createInstance` attribute in the initial `receive` activity. In this initial `receive` activity, the `createInstance` element is set to `yes`. This starts a new

instance of the BPEL process service component. At least one instance startup is required for a conversation. For this reason, you set the `createInstance` variable to `no` in the second `receive` activity.

[Example 8-6](#) shows the source code for the `createInstance` attribute:

Example 8-6 *createInstance* Attribute

```
<!-- receive input from requester -->
<receive name="receiveInput" partnerLink="client"
        portType="tns:LoanBroker"
        operation="initiate" variable="input"
        createInstance="yes"/>
```

8.2.2.7 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. You can also explicitly specify a dehydration point with a `dehydrate` activity. For more information, see [Section A.2.8, "Dehydrate Activity."](#)

8.2.2.8 Multiple Runtime Endpoint Locations

Oracle SOA Suite provides support for specifying multiple partner link endpoint locations. This capability is useful for failover purposes if the first endpoint is down. To provide an alternate partner link endpoint location, add the `location` attribute to the `composite.xml` file. [Example 8-7](#) provides an example.

Example 8-7 *Alternate Runtime Endpoint Location*

```
<reference name="HeaderService ...>
<binding.ws port="http://services.otn.com/HelloWorldApp#wsdl.endpoint(client/
    HelloWorldService_pt)"
location="http://server:port/soa-infra/services/default/
    HelloWorldService!1.0/client?WSDL">
<property name="endpointURI">http://jsmith.us.oracle.com:80/a.jsp
@http://myhost.us.oracle.com:8888/soa-infra/services/HelloWorldApp/HelloWorld!
1.0*2007-10-22_14-33-04_195/client
    </property>
</binding.ws>
</reference>
```

8.2.3 What You May Need to Know About Limitations on BPEL 2.0 IMA Support

Receive activities are a type of inbound message activity (IMA). Other examples of IMAs are as follows:

- onMessage branches of a scope activity (in BPEL 1.1) or a pick activity
- onEvent branches of a scope activity in BPEL 2.0

The BPEL 2.0 specification allows multiple IMAs to work with each other or with other IMAs derived from extension activities. To provide for consistent runtime behavior,

the BPEL 2.0 specification allows for correlation sets with the `initiate` attribute set to `join`.

However, Oracle BPEL Process Manager's implementation of the BPEL 2.0 specification does *not* support this behavior. The only way to support multiple IMAs is by coding them as `onMessage` branches for a `pick` activity (that is, setting `createInstance` to `yes`).

Oracle BPEL Process Manager also does not support other forms of multiple IMAs, such as a flow activity with two branches, each with a receive activity and with `createInstance` set to `yes` and correlation sets with `initiate` set to `join`.

As a workaround, you must design two different BPEL processes with the two receive activities in alternating order, as follows:

- Process1 with `receive1` followed by `receive2`, and only `receive1` having `createInstance` set to `yes`
- Process2 with `receive2` followed by `receive1`, and only `receive2` having `createInstance` set to `yes`.

The same also applies for any other combination of IMAs, such as a receive activity and `pick` activity, or two `pick` activities.

8.2.4 What Happens When You Specify a Conversation ID

You can also enter an optional conversation ID value in the **Conversation ID** field of an `invoke` activity (and other activities such as a receive activity and the `onMessage` branch of a `pick` or `scope` activity).

The conversation ID identifies a process instance during an asynchronous conversation. By default, the BPEL process service engine generates a unique ID for each conversation (which can span multiple `invoke` and receive activities), as specified by WSA addressing. If you want, you can specify your own value for the service engine to use. Conversation IDs are implemented with the `bpelx:conversationId` extension.

8.2.4.1 `bpelx:conversationId` in BPEL 1.1

[Example 8-8](#) provides an example of the `bpelx:conversationId` extension in a BPEL project that supports BPEL version 1.1. The `bpelx:conversationId` extension takes an XPath expression.

Example 8-8 *`bpelx:conversationId` Conversation ID in BPEL 1.1*

```
<invoke ... bpelx:conversationId="$convId2">
</invoke>

<receive ... bpelx:conversationId="$convId2">
</receive>

<onMessage... bpelx:conversationId="$convId2">
</onMessage>
```

8.2.4.2 `bpelx:conversationId` in BPEL 2.0

[Example 8-9](#) provides an example of the `bpelx:conversationId` extension in a BPEL project that supports BPEL version 2.0. The `bpelx:conversationId` extension takes a BPEL 2.0 XPath expression.

Example 8–9 *bpelx:conversationId* Conversation ID in BPEL 2.0

```

<invoke ...>
  <bpelx:conversationId>$convId1</bpelx:conversationId>
</invoke>

<receive ...>
  <bpelx:conversationId>$convId1</bpelx:conversationId>
</receive>

<onMessage ...>
  <bpelx:conversationId>$convId2</bpelx:conversationId>
</onMessage>

```

8.3 Using a Dynamic Partner Link at Runtime

You can dynamically configure a partner link at runtime in BPEL. This is useful for scenarios in which the target service that BPEL wants to invoke is not known until runtime.

Note: Dynamic partner links are only supported in BPEL 1.1 projects.

8.3.1 How To Add and Use a Dynamic Partner Link at Runtime

1. Create a WSDL file that contains multiple services that use the same `portType`.
2. Create a reference binding component entry in the `composite.xml` file that uses the WSDL:

```

<reference name="loanService">
  <interface.wSDL
    interface="http://services.otn.com#wsdl.interface(LoanService)"
    callbackInterface="http://services.otn.com#wsdl.interface(LoanServiceCallback)"
  />
  <binding.ws port=
    "http://services.otn.com#wsdl.endpoint(AmericanLoan/LoanService_pt)"/>
</reference>

```

Notes:

- Adding the `binding.ws port` setting is optional. This is because the port is overridden at runtime by properties passed from BPEL.
 - If there is no `port` setting, and there is no composite import of the concrete WSDL associated with this reference, you must specify the location of the concrete WSDL with a `location` attribute.
-

3. In the BPEL file, programmatically assign the partner link. For this example, `UnitedLoan` is one of the services defined in the WSDL.

```

<copy>
  <from>
    <EndpointReference
      xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
    <Address>http://myhost.us.oracle.com:9700/orabpel/default/UnitedLoan</Address>
    </EndpointReference>

```

```

    </from>
    <to partnerLink="LoanService"/>
  </copy>

```

8.4 Using WS-Addressing in an Asynchronous Service

Because there can be many active instances at any time, the server must be able to direct web service responses to the correct BPEL process service component instance. You can use WS-Addressing to identify asynchronous messages to ensure that asynchronous callbacks locate the appropriate client.

Figure 8–3 provides an overview of WS-Addressing. WS-Addressing uses Simple Object Access Protocol (SOAP) headers for asynchronous message correlation. Messages are independent of the transport or application used.

Figure 8–3 Callback with WS-Addressing Headers

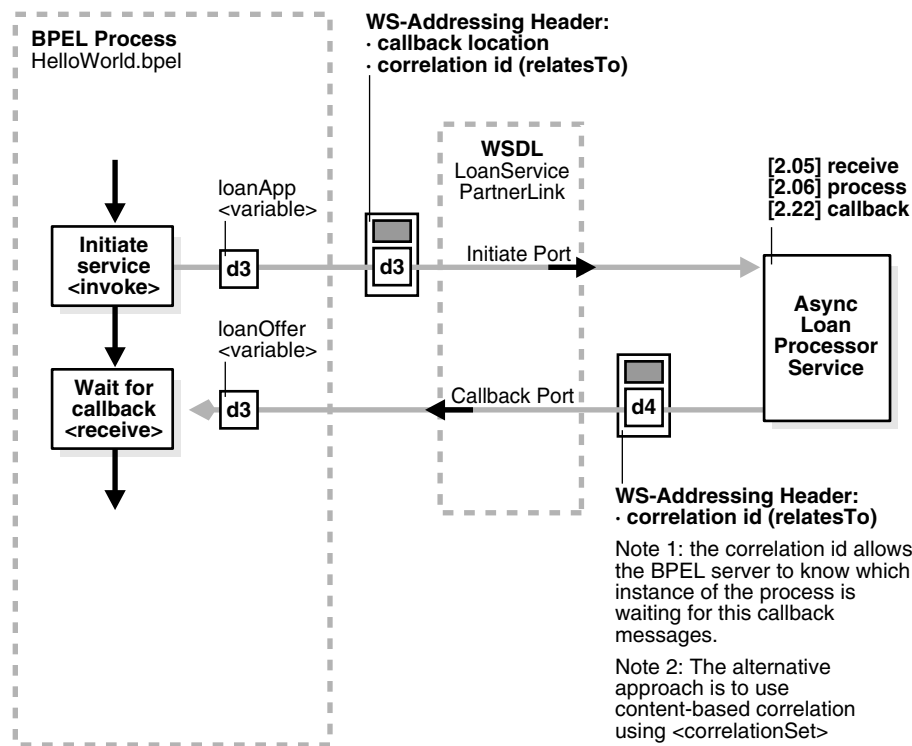


Figure 8–3 shows how messages are passed along with WS headers so that the response can be sent to the correct destination.

The example in this chapter uses WS-Addressing for correlation. To view the messages, you can use TCP tunneling, which is described in [Section 8.4.1.1, "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.

8.4.1 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.

8.4.1.1 Using TCP Tunneling to See Messages Exchanged Between Programs

The messages that are exchanged between programs and services can be seen through TCP tunneling. This is particularly useful when you want to see the exact SOAP messages exchanged between the BPEL process service component flow and web services.

To monitor the SOAP messages, insert a software listener between your flow and the service. Your flow communicates with the listener (called a TCP tunnel) and the listener forwards your messages to the service, and displays them. Likewise, responses from the service are returned to the tunnel, which displays them and then forwards them back to the flow.

To see all the messages exchanged between the server and a web service, you need only a single TCP tunnel for synchronous services because all the pertinent messages are communicated in a single request and reply interaction with the service. For asynchronous services, you must set up two tunnels, one for the invocation of the service and another for the callback port of the flow.

8.4.1.1.1 Setting Up a TCP Listener for Synchronous Services Follow these steps to set up a TCP listener for synchronous services initiated by an Oracle BPEL Process Manager process:

1. Visit the following URL for instructions on how to download and install Axis TCP Monitor (`tcpmon`)

<http://ws.apache.org/commons/tcpmon/>

2. Visit the following URL for instructions on how to use `tcpmon`:

<http://ws.apache.org/axis/java/user-guide.html>

3. Place `axis.jar` in your class path.

4. Start `tcpmon`:

```
C:\...\> java org.apache.axis.utils.tcpmon localport remoteHost
port_on_which_remote_server_is_running
```

5. In the `composite.xml` file, add the `endpointURI` property under `binding.ws` for your flow to override the endpoint of the service.

6. From the operating system command prompt, compile and deploy the process with ant.

Note that the same technique can 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.

8.4.1.1.2 Setting Up a TCP Listener for Asynchronous Services Follow these steps to set up a TCP listener to display the SOAP messages for callbacks from asynchronous services:

1. Start a TCP listener to listen on a port and to send the Oracle BPEL Process Manager port.
 - a. Open Oracle Enterprise Manager Fusion Middleware Control.
 - b. From the **SOA Infrastructure** menu, select **SOA Administration > Common Properties**.
 - c. Specify the value for **Callback Server URL**. This URL is sent by the server as part of the asynchronous callback address to the invoker.
2. From the **SOA Infrastructure** menu, select **Administration > System MBean Browser**.
3. Expand **Application Defined MBeans > oracle.soa.config > Server : soa_server > SCAComposite**.

where *soa_server* is the specific server instance name (for example, **AdminServer**).

All the SOA composite applications deployed on the server appear.

4. Follow these steps to set this property on a composite application. This action enables it to apply to all bindings in the composite application.
 - a. Click your composite.
 - b. Ensure the **Attributes** tab is selected.
 - c. In the **Name** column, click **Properties**.
 - d. Click the **Add** icon.
 - e. Expand the newly added **Element_number** (appears at the end of the list).

where *number* is the next sequential number beyond the last property. For example, if the property list contains twelve elements, adding a new property causes **Element_13** to be displayed.
 - f. In the **name** field, enter `oracle.webservices.local.optimization`.
 - g. In the **value** field, enter `false`.
 - h. In the **many** field, enter `false`.
 - i. Click **Apply**, and then click **Return**.
 - j. In the **Name** column on the **Operations** tab, click **save**.
 - k. Click **Invoke** to execute the operation.
 - l. Click **Return** or click a node in the **System MBean Browser** pane.

Note: After adding, deleting, or updating a property, you can click the **Refresh cached tree data** icon in the upper right corner of the System MBean Browser page to see the new data.

5. Follow these steps to set this property on a specific binding.
 - a. Expand your composite application, and drill down to the specific **SCAComposite.SCAResource.SCABinding** folder.
 - b. Click **WSBinding**.
 - c. Perform steps 4b through 4l.
6. Initiate any flow that invokes asynchronous web services. You can combine this with the synchronous TCP tunneling configuration to send a service initiation request through your first TCP tunnel.

The callbacks from the asynchronous services are shown in the TCP listener.

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.

8.5 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.

8.5.1 How to Use Correlation Sets in an Asynchronous Service

This section describes the steps to perform to use correlation sets in an asynchronous service.

8.5.1.1 Step 1: Creating a Project

To create a project:

1. Start Oracle JDeveloper.
2. From the **File** main menu, select **New > Applications**.
3. Select **SOA Application**, and click **OK**.

The Create SOA Application Wizard appears.

4. In the **Application Name** field, enter `MyCorrelationSetApp`.
5. Accept the default values for all remaining settings, and click **Next**.
6. In the **Project Name** field, enter `MyCorrelationSetComposite`.
7. Accept the default values for all remaining settings, and click **Next**.

8. In the **Composite Template** section, select **Composite With BPEL Process**, and click **Finish**.

The Create BPEL Process dialog appears.

9. Enter the values shown in [Table 8–1](#).

Table 8–1 Create BPEL Process Dialog Fields and Values

Field	Value
Name	Enter MyCorrelationSet.
Template	Select Asynchronous BPEL Process .
Expose as a SOAP Service	Select the checkbox. After process creation, note the SOAP service that appears in the Exposed Services swimlane. This service provides the entry point to the composite application from the outside world.

10. Accept the default values for all remaining settings, and click **OK**.

8.5.1.2 Step 2: Configuring Partner Links and File Adapter Services

You now create three partner links that use the SOAP service.

This section contains these topics:

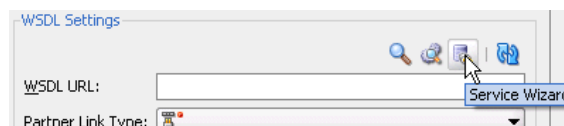
- You create an initial partner link with an adapter service for reading a loan application.
- You create a second partner link with an adapter service for reading an application response.
- You create a third partner link with an adapter service for reading a customer response.

8.5.1.2.1 Creating an Initial Partner Link and File Adapter Service

To create an initial partner link and file adapter service:

1. Double-click the **MyCorrelationSet** BPEL process.
2. In the Component Palette, expand **BPEL Constructs**.
3. Drag an initial **Partner Link** activity into the right swimlane of the designer.
4. Click the third icon at the top (the **Service Wizard** icon). This starts the Adapter Configuration Wizard, as shown in [Figure 8–4](#).

Figure 8–4 Adapter Configuration Wizard Startup



5. In the Configure Service or Adapter dialog, select **File Adapter** and click **OK**.
6. In the Welcome dialog, click **Next**.
7. In the **Service Name** field of the Service Name dialog, enter `FirstReceive` and click **Next**.
8. In the Adapter Interface dialog, accept the default settings and click **Next**.

9. In the Operation dialog, select **Read File** as the **Operation Type** and click **Next**. The **Operation Name** field is automatically filled in with **Read**.
10. Above the **Directory for Incoming Files (physical path)** field, click **Browse**.
11. Select a directory from which to read files (for this example, **C:\files\receiveprocess\FirstInputDir** is selected).
12. Click **Select**.
13. Click **Next**.
14. In the File Filtering dialog, enter appropriate file filtering parameters.
15. Click **Next**.
16. In the File Polling dialog, enter appropriate file polling parameters.
17. Click **Next**.
18. In the Messages dialog, click **Browse** next to the **URL** field to display the Type Chooser dialog.
19. Select an appropriate XSD schema file. For this example, **Book1_4.xsd** is the schema and **LoanAppl** is the schema element selected.
20. Click **OK**.
The **URL** field (**Book1_4.xsd** for this example) and the **Schema Element** field (**LoanAppl** for this example) are filled in.
21. Click **Next**.
22. Click **Finish**.

You are returned to the Partner Link dialog. All other fields are automatically completed. The dialog looks as shown in [Table 8-2](#):

Table 8-2 Partner Link Dialog Fields and Values

Field	Value
Name	FirstReceive
WSDL URL	<i>directory_path</i> /FirstReceive.wsdl
Partner Link Type	Read_plt
Partner Role	Leave unspecified.
My Role	Read_role

23. Click **OK**.

8.5.1.2.2 Creating a Second Partner Link and File Adapter Service

To create a second partner link and file adapter service:

1. Drag a second **PartnerLink** activity beneath the **FirstReceive** partner link activity.
2. At the top, click the third icon (the **Service Wizard** icon).
3. In the Configure Service or Adapter dialog, select **File Adapter** and click **OK**.
4. In the Welcome dialog, click **Next**.
5. In the Adapter Type dialog, select **File Adapter** and click **Next**.

6. In the **Service Name** field of the Service Name dialog, enter `SecondFileRead` and click **Next**. This name must be unique from the one you entered in Step 7 of [Section 8.5.1.2.1, "Creating an Initial Partner Link and File Adapter Service."](#)
7. In the Adapter Interface dialog, accept the default settings and click **Next**.
8. In the Operation dialog, select **Read File** as the **Operation Type**.
9. In the **Operation Name** field, change the name to `Read1`.
10. Click **Next**.
11. Select **Directory Names are Specified as Physical Path**.
12. Above the **Directory for Incoming Files (physical path)** field, click **Browse**.
13. Select a directory from which to read files (for this example, `C:\files\receiveprocess\SecondInputDir` is entered).
14. Click **Select**.
15. Click **Next**.
16. Enter appropriate file filtering parameters in the File Filtering dialog.
17. Click **Next**.
18. Enter appropriate file polling parameters in the File Polling dialog.
19. Click **Next**.
20. Next to the **URL** field in the Messages dialog, click **Browse** to display the Type Chooser dialog.
21. Select an appropriate XSD schema file. For this example, **Book1_5.xsd** is the schema and **LoanAppResponse** is the schema element selected.
22. Click **OK**.
The **URL** field (**Book1_5.xsd** for this example) and the **Schema Element** field (**LoanAppResponse** for this example) are filled in.
23. Click **Next**.
24. Click **Finish**.

You are returned to the Partner Link dialog. All other fields are automatically completed. The dialog looks as shown in [Table 8-3](#):

Table 8-3 Partner Link Dialog Fields and Values

Field	Value
Name	SecondReceive
WSDL URL	<i>directory_path/SecondFileRead.wsdl</i>
Partner Link Type	Read1_plt
Partner Role	Leave unspecified.
My Role	Read1_role

25. Click **OK**.

8.5.1.2.3 Creating a Third Partner Link and File Adapter Service

To create a third partner link and file adapter service:

1. Drag a third **PartnerLink** activity beneath the **SecondReceive partner link** activity.
2. At the top, click the third icon (the **Service Wizard** icon).
3. In the Configure Service or Adapter dialog, select **File Adapter** and click **OK**.
4. In the Welcome dialog, click **Next**.
5. In the Adapter Type dialog, select **File Adapter** and click **Next**.
6. In the **Service Name** field of the Service Name dialog, enter `ThirdFileRead` and click **Next**. This name must be unique from the one you entered in Step 7 on page 8-16 and Step 6 of [Section 8.5.1.2.2, "Creating a Second Partner Link and File Adapter Service."](#)
7. In the Adapter Interface dialog, accept the default settings and click **Next**.
8. In the Operation dialog, select **Read File** as the **Operation Type**.
9. In the **Operation Name** field, change the name to `Read2`. This name must be unique.
10. Click **Next**.
11. Select **Directory Names are Specified as Physical Path**.
12. Above the **Directory for Incoming Files (physical path)** field, click **Browse**.
13. Select a directory from which to read files (for this example, `C:\files\receiveprocess\ThirdInputDir` is entered).
14. Click **Select**.
15. Click **Next**.
16. Enter appropriate file filtering parameters in the File Filtering dialog.
17. Click **Next**.
18. Enter appropriate file polling parameters in the File Polling dialog.
19. Click **Next**.
20. Next to the **URL** field in the Messages dialog, click **Browse** to display the Type Chooser dialog.
21. Select an appropriate XSD schema file. For this example, **Book1_6.xsd** is the schema and **CustResponse** is the schema element selected.
22. Click **OK**.
The **URL** field (**Book1_6.xsd** for this example) and the **Schema Element** field (**CustResponse** for this example) are filled in.
23. Click **Next**.
24. Click **Finish**.

You are returned to the Partner Link dialog. All other fields are automatically completed. The dialog looks as shown in [Table 8-4](#):

Table 8-4 Partner Link Dialog Fields and Values

Field	Value
Name	ThirdReceive

Table 8–4 (Cont.) Partner Link Dialog Fields and Values

Field	Value
WSDL URL	<i>directory_path/ThirdFileRead.wsdl</i>
Partner Link Type	Read2_plt
Partner Role	Leave unspecified.
My Role	Read2_role

25. Click OK.

8.5.1.3 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.

8.5.1.3.1 Creating an Initial Receive Activity

To create an initial receive activity:

1. Expand **BPEL Constructs** in the Component Palette.
2. Drag a **Receive** activity beneath the **receiveInput** receive activity in the designer.
3. Double-click the **receive** icon to display the Receive dialog.
4. Enter the details described in [Table 8–5](#) to associate the first partner link (**FirstReceive**) with the first receive activity:

Table 8–5 Receive Dialog Fields and Values

Field	Value
Name	receiveFirst
Partner Link	FirstReceive
Create Instance	Select this checkbox.

The **Operation (Read)** field is automatically filled in.

5. To the right of the **Variable** field, click the first icon. This is the automatic variable creation icon.
6. In the Create Variable dialog, click OK.
A variable named **receiveFirst_Read_InputVariable** is automatically created in the **Variable** field.
7. Ensure that you selected the **Create Instance** checkbox, as mentioned in Step 4.
8. Click OK.

8.5.1.3.2 Creating a Second Receive Activity

To create a second receive activity:

1. From the Component Palette, drag a second **Receive** activity beneath the **receiveFirst** receive activity.
2. Double-click the **receive** icon to display the Receive dialog.

3. Enter the details described in [Table 8–6](#) to associate the second partner link (**SecondReceive**) with the second receive activity:

Table 8–6 Receive Dialog Fields and Values

Field	Value
Name	receiveSecond
Partner Link	SecondFileRead
Create Instance	Do <i>not</i> select this checkbox.

The **Operation** (**Read1**) field is automatically filled in.

4. To the right of the **Variable** field, click the first icon.
5. In the Create Variable dialog, click **OK**.

A variable named **receiveSecond_Read1_InputVariable** is automatically created in the **Variable** field.

6. Click **OK**.

8.5.1.3.3 Creating a Third Receive Activity

To create a third receive activity:

1. From the Component Palette, drag a third **Receive** activity beneath the **receiveSecond** receive activity.
2. Double-click the **receive** icon to display the Receive dialog.
3. Enter the details described in [Table 8–7](#) to associate the third partner link (**ThirdReceive**) with the third receive activity:

Table 8–7 Receive Dialog Fields and Values

Field	Value
Name	receiveThird
Partner Link	ThirdFileRead
Create Instance	Do <i>not</i> select this checkbox.

The **Operation** (**Read2**) field is automatically filled in.

4. To the right of the **Variable** field, click the first icon.
5. In the Create Variable dialog, click **OK**.

A variable named **receiveThird_Read2_InputVariable** is automatically created in the **Variable** field.

6. Click **OK**.

Each receive activity is now associated with a specific partner link.

8.5.1.4 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.

8.5.1.4.1 Creating an Initial Correlation Set

To create an initial correlation set:

1. In the Structure window of Oracle JDeveloper, right-click **Correlation Sets** and select **Expand All Child Nodes**.
2. In the second **Correlation Sets** folder, right-click and select **Create Correlation Set**.
3. In the **Name** field of the Create Correlation Set dialog, enter `CorrelationSet1`.
4. In the **Properties** section, click the **Add** icon to display the Property Chooser dialog.
5. Select **Properties**, then click the **Add** icon (first icon at the top) to display the Create Property dialog.
6. In the **Name** field, enter `NameCorr`.
7. To the right of the **Type** field, click the **Browse** icon.
8. In the Type Chooser dialog, select **string** and click **OK**.
9. Click **OK** to close the Create Property dialog, the Property Chooser dialog, and the Create Correlation Set dialog.

8.5.1.4.2 Creating a Second Correlation Set**To create a second correlation set:**

1. Return to the **Correlation Sets** section in the Structure window of Oracle JDeveloper.
2. Right-click the **Correlation Sets** folder and select **Create Correlation Set**.
3. In the **Name** field of the Create Correlation Set dialog, enter `CorrelationSet2`.
4. In the **Properties** section, click the **Add** icon to display the Property Chooser dialog.
5. Select **Properties**, then click the **Add** icon to display the Create Property dialog.
6. In the **Name** field, enter `IDCorr`.
7. To the right of the **Type** field, click the **Browse** icon.
8. In the Type Chooser dialog, select **double** and click **OK**.
9. Click **OK** to close the Create Property dialog, the Property Chooser dialog, and the Create Correlation Set dialog.

8.5.1.5 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.

8.5.1.5.1 Associating the First Correlation Set with a Receive Activity**To associate the first correlation set with a receive activity:**

1. Double-click the **receiveFirst** receive activity to display the Receive dialog.
2. Click the **Correlations** tab.

3. Click the **Add** icon to display the correlation set dropdown list.
4. Select **CorrelationSet1**.
5. Click the **Initiate** column to display a dropdown list, and select **yes**. When set to **yes**, the set is initiated with the values of the properties occurring in the message being exchanged.
6. Click **OK**.

8.5.1.5.2 Associating the Second Correlation Set with a Receive Activity

To associate the second correlation set with a receive activity:

1. Double-click the **receiveSecond** receive activity to display the Receive dialog.
2. Click the **Correlations** tab.
3. Click the **Add** icon to display the correlation set dropdown list.
4. Select **CorrelationSet2**, then click **OK**.
5. Click the **Initiate** column to display a dropdown list, and select **yes**.
6. Click **Add** again and select **CorrelationSet1**.
7. Click **OK**.
8. Click the **Initiate** column to display a dropdown list, and select **no** for **CorrelationSet1**.
9. Click **OK**.

This groups the first and second receive activities into a correlated group.

8.5.1.5.3 Associating the Third Correlation Set with a Receive Activity

To associate the third correlation set with a receive activity:

1. Double-click the **receiveThird** receive activity to display the Receive dialog.
2. Click the **Correlations** tab.
3. Click the **Add** icon.
4. Select **CorrelationSet2**.
5. Set the **Initiate** column to **no** for **CorrelationSet2**.
6. Click **OK**.

This groups the second and third receive activities into a second correlated group.

8.5.1.6 Step 6: Creating Property Aliases

Property aliases enable you to map a global property to a field in a specific message part. This action enables the property name to become an alias for the message part and location. The alias can be used in XPath expressions.

8.5.1.6.1 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 **FirstReceive** 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 **SecondReceive** partner link (defined by the **SecondFileRead.wsdl** file).

To create property aliases for NameCorr:

1. In the Structure window of Oracle JDeveloper, right-click **Property Aliases**.
2. Select **Create Property Alias**.
3. From the **Property** list, select **NameCorr**.
4. Expand and select **Message Types > Partner Link > FirstReceive > FirstReceive.wsdl > Message Types > LoanApp1_msg > Part - LoanApp1**.
5. In the **Query** field, press Ctrl+Space to define the following XPath expression:
`/ns2:LoanApp1/ns2:Name`
6. Click **OK**.
7. Repeat Step 1 through Step 3 to create a second property alias for **NameCorr**.
8. Expand and select **Message Types > Project WSDL Files > SecondFileRead.wsdl > Message Types > LoanAppResponse_msg > Part - LoanAppResponse**.
9. In the **Query** field, press Ctrl+Space to define the following XPath expression:
`/ns4:LoanAppResponse/ns4:APR`
10. Click **OK**.

8.5.1.6.2 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 **SecondReceive** 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 **ThirdReceive** partner link (defined by the **ThirdFileRead.wsdl** file).

To create property aliases for IDCorr:

1. In the Structure window, right-click **Property Aliases**.
2. Select **Create Property Alias**.
3. In the **Property** list, select **IDCorr**.
4. Expand and select **Message Types > Project WSDL Files > SecondFileRead.wsdl > Message Types > LoanAppResponse_msg > Part - LoanAppResponse**.
5. In the **Query** field, press Ctrl+Space to define the following XPath expression:
`/ns4:LoanAppResponse/ns4:APR`
6. Click **OK**.
7. Repeat Step 1 through Step 3 to create a second property alias for **IDCorr**.
8. Expand and select **Message Types > Project WSDL Files > ThirdFileRead.wsdl > Message Types > CustResponse_msg > Part - CustResponse**.
9. In the **Query** field, press Ctrl+Space to define the following XPath expression:

```
/ns6:CustResponse/ns6:APR
```

Design is now complete.

10. Click OK.

8.5.1.7 Step 7: Reviewing WSDL File Content

To review WSDL file content:

1. Refresh the Application Navigator.

The NameCorr and IDCorr correlation set properties are defined in the `MyCorrelationSet_Properties.wsdl` file in the Application Navigator. [Example 8–10](#) provides an example.

Example 8–10 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 8–11](#):

Example 8–11 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 8–12](#) provides an example.

Example 8–12 WSDL File Import

```
<import namespace="http://xmlns.oracle.com/MyCorrelationSet"
```

```
location="MyCorrelationSet.wsdl"/>
```

8.5.2 What You May Need to Know About Setting Correlations for an IMA Using a `fromParts` Element With Multiple Parts

Assume you have the following scenario:

- A BPEL 2.0 process with a WSDL message type that has multiple parts that are identical in type.
- A property alias has been defined based on the element type of the above part.

For a process that has an inbound message activity (IMA) (for example, a receive activity, `onMessage` branch of a scope or pick activity, or `onEvent` branch of a scope activity in BPEL 2.0) that uses the `fromParts` element with `fromPart` defined for each part, correlations cannot be defined because the runtime environment cannot determine the part to which to apply the property alias.

For more information about mapping WSDL message parts with the `toParts` and `fromParts` elements, see [Section 6.17, "Mapping WSDL Message Parts in BPEL 2.0."](#)

Using Parallel Flow in a BPEL Process

This chapter describes how to use parallel flow in a BPEL process service component. Parallel flows enable a BPEL process service component to perform multiple tasks at the same time. Parallel flow is especially useful when you must perform several time-consuming and independent tasks.

This chapter includes the following sections:

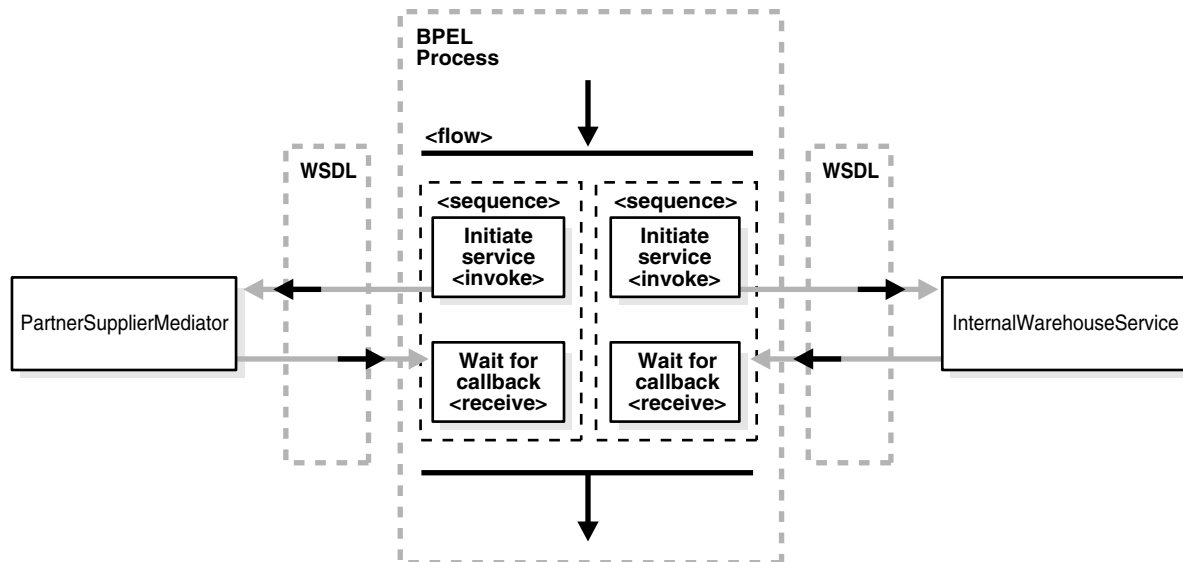
- [Section 9.1, "Introduction to Parallel Flows in BPEL Processes"](#)
- [Section 9.2, "Creating a Parallel Flow"](#)
- [Section 9.3, "Customizing the Number of Parallel Branches"](#)

For additional information on creating parallel flows in a SOA composite application, see the Fusion Order Demo application, which is described in [Chapter 3, "Introduction to the SOA Sample Application."](#)

9.1 Introduction to Parallel Flows in BPEL Processes

A BPEL process service component must sometimes gather information from multiple asynchronous sources. Because each callback can take an undefined amount of time (hours or days), it may take too long to call each service one at a time. By breaking the calls into a parallel flow, a BPEL process service component can invoke multiple web services at the same time, and receive the responses as they come in. This method is much more time efficient.

[Figure 9-1](#) shows the `Retrieve_QuotesFromSuppliers` flow activity of the Fusion Order Demo application. The `Retrieve_QuotesFromSuppliers` flow activity sends order information to two suppliers in parallel: an internal warehouse (`InternalWarehouseService`) and an external partner warehouse (`PartnerSupplierMediator`). The two warehouses return their bids for the order to the flow activity. Here, two asynchronous callbacks execute in parallel. One callback does not have to wait for the other to complete first. Each response is stored in a different global variable.

Figure 9–1 Parallel Flow Invocation

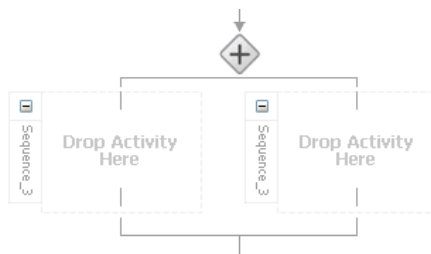
9.2 Creating a Parallel Flow

You can create a parallel flow in a BPEL process service component with the flow activity. The flow activity enables you to specify one or more activities to be performed concurrently. The flow activity also provides synchronization. The flow activity completes when all activities in the flow have finished processing. Completion of this activity includes the possibility that it can be skipped if its enabling condition is false.

9.2.1 How to Create a Parallel Flow

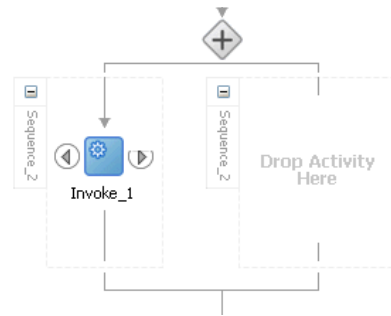
To create a parallel flow:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag a **Flow** activity into the designer.
3. Click the + sign to expand the flow activity, as shown in [Figure 9–2](#).

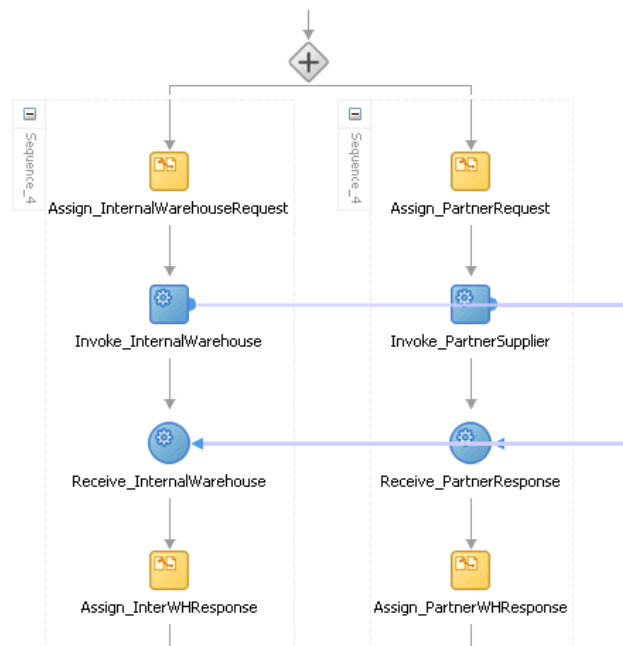
Figure 9–2 Flow Activity

The flow activity initially includes two branches, each with a box for functional elements. Populate these boxes as you do a scope activity, either by building a function or dragging activities into the boxes. You can add additional branches by highlighting the flow activity and clicking the **Add Sequence** icon.

4. Drag and define additional activities onto each side of the flow to invoke multiple services at the same time. [Figure 9–3](#) provides details.

Figure 9–3 Expanded Flow Activity

When complete, flow activity design can look as shown in [Figure 9–4](#). This example shows the **Retrieve_QuotesFromSuppliers** flow activity of the Fusion Order Demo application. Two branches are defined for receiving bids: one for **InternalWarehouseService** and the other for **PartnerSupplierMediator**.

Figure 9–4 Flow Activity After Design Completion

9.2.2 What Happens When You Create a Parallel Flow

A flow activity typically contains many sequence activities. Each sequence is performed in parallel. [Example 9–1](#) shows the syntax for two sequences of the **Retrieve_QuotesFromSuppliers** flow activity in the `OrderProcessor.bpel` file after design completion. However, a flow activity can have many sequences. A flow activity can also contain other activities. In [Example 9–1](#), each sequence in the flow contains assign, invoke, and receive activities.

Example 9–1 Flow Activity

```
<flow name="Retrieve_QuotesFromSuppliers">
  <sequence name="Sequence_4">
    <assign name="Assign_InternalWarehouseRequest">
      <copy>
        <from variable="gOrderInfoVariable"
```

```

        query="/ns4:orderInfoVOSDO/ns4:OrderId"/>
        <to variable="lInternalWarehouseInputVariable"
            part="payload"
            query="/ns1:WarehouseRequest/ns1:orderId"/>
    </copy>
</assign>
<invoke name="Invoke_InternalWarehouse"
    inputVariable="lInternalWarehouseInputVariable"
    partnerLink="InternalWarehouseService"
    portType="ns1:InternalWarehouseService"
    operation="process"/>
<receive name="Receive_InternalWarehouse"
    createInstance="no"
    variable="lInternalWarehouseResponseVariable"
    partnerLink="InternalWarehouseService"
    portType="ns1:InternalWarehouseServiceCallback"
    operation="processResponse"/>
<assign name="Assign_InterWHResponse">
    <bpelx:append>
        <bpelx:from variable="lInternalWarehouseResponseVariable"
            part="payload"
            query="/ns1:WarehouseResponse"/>
        <bpelx:to variable="gWarehouseQuotes"
            query="/ns1:WarehouseList"/>
    </bpelx:append>
</assign>
</sequence>
<sequence name="Sequence_4">
    <assign name="Assign_PartnerRequest">
        <copy>
            <from variable="gOrderInfoVariable"
                query="/ns4:orderInfoVOSDO"/>
            <to variable="lPartnerSupplierInputVariable"
                part="request" query="/ns4:orderInfoVOSDO"/>
        </copy>
    </assign>
    <invoke name="Invoke_PartnerSupplier"
        partnerLink="PartnerSupplierMediator"
        portType="ns15:execute_ptt" operation="execute"
        inputVariable="lPartnerSupplierInputVariable"/>
    <receive name="Receive_PartnerResponse"
        createInstance="no"
        variable="lPartnerResponseVariable"
        partnerLink="PartnerSupplierMediator"
        portType="ns15:callback_ptt" operation="callback"/>
    <assign name="Assign_PartnerWHResponse">
        <bpelx:append>
            <bpelx:from variable="lPartnerResponseVariable"
                part="callback"
                query="/ns1:WarehouseResponse"/>
            <bpelx:to variable="gWarehouseQuotes"
                query="/ns1:WarehouseList"/>
        </bpelx:append>
    </assign>
</sequence>
</flow>

```

9.2.3 Synchronizing the Execution of Activities in a Flow Activity

You can synchronize the execution of activities within a flow activity to ensure that certain activities only execute after other activities have completed. For example, assume you have an invoke activity, `verifyFlight`, that is executed in parallel with other invoke activities (`verifyHotel`, `verifyCarRental`, and `scheduleFlight`) when the flow activity begins. However, scheduling a flight is necessary only after verifying that a flight is available. Therefore, you can add a link between the `verifyFlight` and `scheduleFlight` invoke activities. Links provide a level of dependency indicating that the activity that is the target of the link (`scheduleFlight`) is only executed if the activity that is the source of the link (`verifyFlight`) has completed.

[Example 9-2](#) provides details. The link name `verifyFlight-To-scheduleFlight` is assigned to the source `verifyFlight` and target `scheduleFlight` invoke activities. If the source `verifyFlight` completes execution, the target `scheduleFlight` is then executed.

Example 9-2 Link Between Source and Target Activities

```
<flow ...>
  <links>
    <link name="verifyFlight-To-scheduleFlight" />
  </links>
  <documentation>
    Verify the availability of a flight, hotel, and rental car in parallel
  </documentation>
  <invoke name="verifyFlight" ...>
    <sources>
      <source linkName="verifyFlight-To-scheduleFlight" />
    </sources>
  </invoke>
  <invoke name="verifyHotel" ... />
  <invoke name="verifyCarRental" ... />
  <invoke name="scheduleFlight" ...>
    <targets>
      <target linkName="verifyFlight-To-scheduleFlight" />
    </targets>
  </invoke>
</flow>
```

[Example 9-2](#) provides an example of link syntax in BPEL version 2.0. The link syntax between BPEL version 1.1 and BPEL version 2.0 is slightly different.

- BPEL version 1.1 uses `<target>` and `<source>`.
- BPEL version 2.0 uses `<targets>` or `<sources>`.

[Table 9-1](#) provides details.

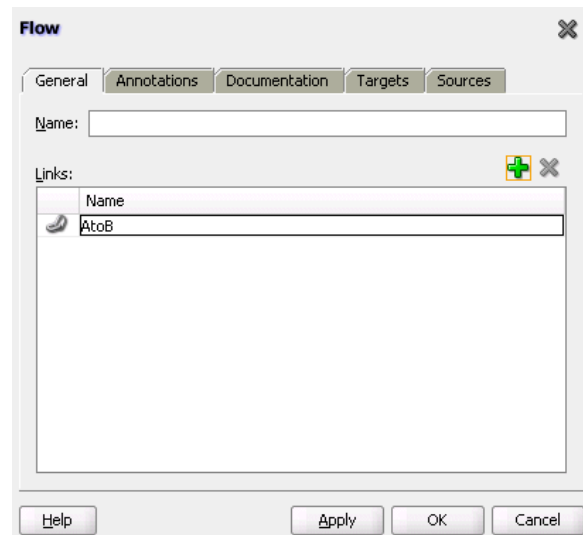
Table 9–1 Links Syntax in BPEL Version 1.1 and BPEL Version 2.0

BPEL Version 1.1 Example	BPEL Version 2.0 Example
<pre> <flow> <links> <link name="XtoY" /> <link name="CtoD" /> </links> <sequence name="X"> <source linkName="XtoY" /> <invoke name="A" .../> <invoke name="B" .../> </sequence> <sequence name="Y"> <target linkName="XtoY" /> <receive name="C" ...> <source linkName="CtoD" /> </receive> <invoke name="E" .../> </sequence> <invoke partnerLink="D" ...> <target linkName="CtoD" /> </invoke> </flow> </pre>	<pre> <flow> <links> <link name="AtoB" /> </links> <assign name="B"> <targets> <target linkName="AtoB" /> </targets> <copy> <from>concat(\$output.payload, 'B')</from> <to>\$output.payload</to> </copy> </assign> <assign name="A"> <sources> <source linkName="AtoB" /> </sources> <copy> <from>concat(\$output.payload, 'A')</from> <to>\$output.payload</to> </copy> </assign> </flow> </pre>

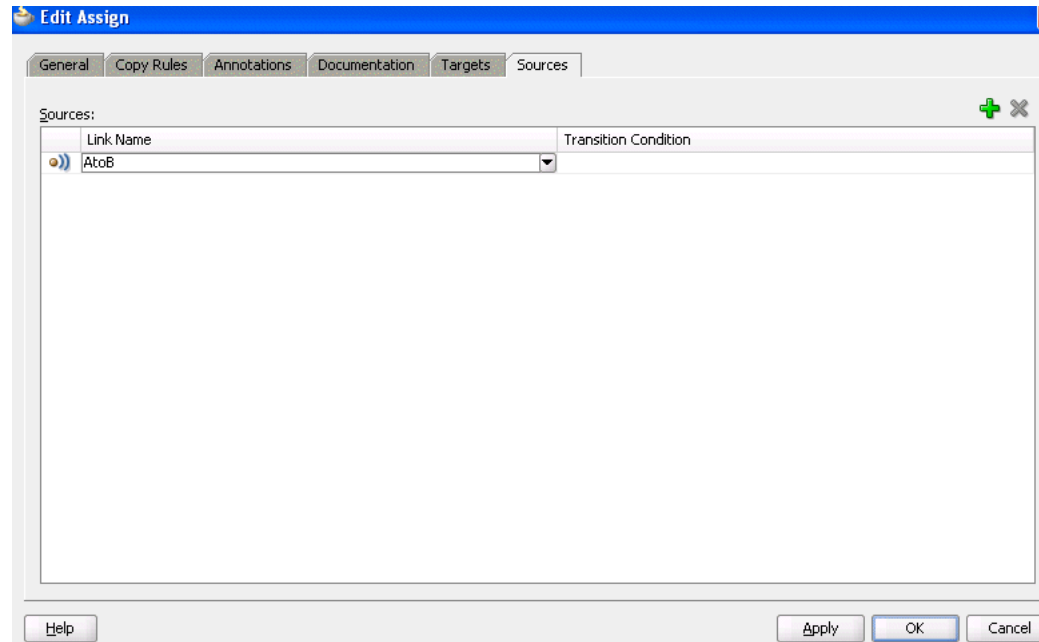
9.2.4 How to Create Synchronization Between Activities Within a Flow Activity

To create synchronization between activities within a flow activity:

1. Create a flow activity. For information, see [Section 9.2.1, "How to Create a Parallel Flow."](#)
2. In the **General** tab of the **Flow** activity, click the **Add** icon.
3. Enter a name for the link, as shown in [Figure 9–5](#).

Figure 9–5 Link Name Creation

4. Click **Apply**, then **OK**.
5. Drag appropriate activities into the flow activity to define as the source with the same link name as defined in Step 3. The value of the link name of the source and target must be the same as the link name declared in the flow activity. For this example, an assign activity named A is defined as the source in [Figure 9–6](#).

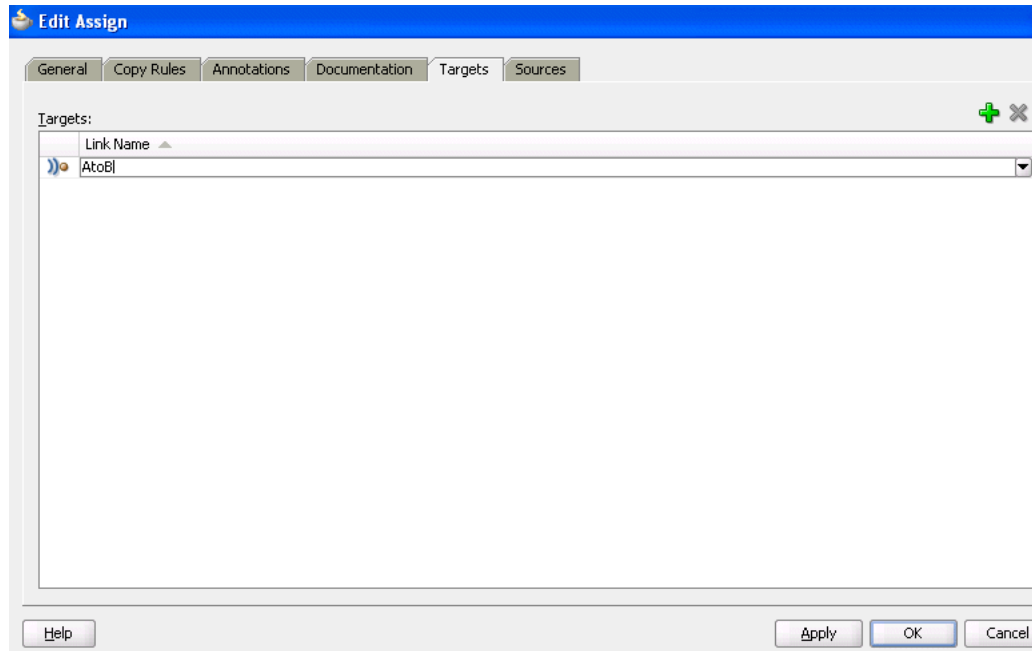
Figure 9–6 Source Activity

Note that each source activity can specify an optional **Transition Condition** as a safe guard for following the specified link. Click the row in this column to invoke the **Browser** icon for accessing the Expression Builder dialog for creating a condition. If the **Transition Condition** column is left blank, it is assumed to evaluate to true.

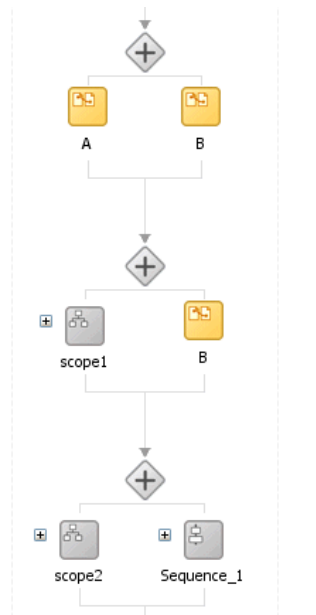
6. Define appropriate copy rules for the assign activity.

7. Click **Apply**, then **OK**.
8. Drag an additional activity into the flow activity to define as the target with the same link name as defined in Step 3. For this example, another assign activity named **B** is defined as the target in [Figure 9-7](#).

Figure 9-7 Target Activity



9. Define appropriate copy rules for the assign activity.
10. Click **Apply**, then **OK**.
11. Continue design of your BPEL process.
When complete, design can appear similar to that shown in [Figure 9-8](#).

Figure 9–8 Three Flow Activities Synchronized with Links

9.2.5 What Happens When You Create Synchronization Between Activities Within a Flow Activity

[Example 9–3](#) shows the .bpel file after design is complete for three flow activities with links for synchronizing activity execution.

- **Flow_1** shows a link between simple activities.
Flow_1 includes a link named `AtoB`. The activity that is the target of the link, assign activity B, is only executed if the activity that is the source of the link, assign activity A, has completed.
- **Flow_2** shows a link between simple activity and composite activity.
Flow_2 also includes the link named `AtoB`. The activity that is the target of the link, assign activity B, is only executed if the activity that is the source of the link, scope activity `scope1`, has completed.
- **Flow_3** shows a link between composite activities.
Flow_3 also includes the link named `AtoB`. The activity that is the target of the link, sequence activity `Sequence_1`, is only executed if the activity that is the source of the link, scope activity `scope2`, has completed.

Example 9–3 Flow Activities with Links

```
<!-- link between simple activities -->
<flow name=Flow_1>
  <links>
    <link name="AtoB"/>
  </links>
  <assign name="A">
    <sources>
      <source linkName="AtoB"/>
    </sources>
  </assign>
  <copy>
    <from>concat($output.payload, 'A')</from>
    <to>$output.payload</to>
  </copy>
</flow>
```

```
        </copy>
      </assign>
    <assign name="B">
      <targets>
        <target linkName="AtoB" />
      </targets>
      <copy>
        <from>concat($output.payload, 'B')</from>
        <to>$output.payload</to>
      </copy>
    </assign>
  </flow>

  <!-- link between simple activity and composite activity -->
  <flow name=Flow_2>
    <links>
      <link name="AtoB" />
    </links>
    <scope name="scope1">
      <sources>
        <source linkName="AtoB" />
      </sources>
      <assign name="A">
        <copy>
          <from>concat($output.payload, 'A')</from>
          <to>$output.payload</to>
        </copy>
      </assign>
    </scope>
    <assign name="B">
      <targets>
        <target linkName="AtoB" />
      </targets>
      <copy>
        <from>concat($output.payload, 'B')</from>
        <to>$output.payload</to>
      </copy>
    </assign>
  </flow>

  <!-- link between composite activities -->
  <flow name=Flow_3>
    <links>
      <link name="AtoB" />
    </links>
    <scope name="scope2">
      <sources>
        <source linkName="AtoB" />
      </sources>
      <assign name="A">
        <copy>
          <from>concat($output.payload, 'A')</from>
          <to>$output.payload</to>
        </copy>
      </assign>
    </scope>
    <sequence name="Sequence_1">
      <targets>
        <target linkName="AtoB" />
      </targets>
```



```

        <assign name="B">
            <copy>
                <from>concat($output.payload, 'B')</from>
                <to>$output.payload</to>
            </copy>
        </assign>
    </sequence>
</flow>
</sequence>

```

9.2.6 What You May Need to Know About Join Conditions in Target Activities

You can specify an optional join condition in target activities. The value of the join condition is a boolean expression. If a join condition is not specified, the join condition is the disjunction (that is, a logical OR operation) of the link status of all incoming links of this activity.

Oracle BPEL Designer does not provide design support for adding join conditions. To add a join condition, you must manually add the condition to the .bpel file in **Source** view in Oracle BPEL Designer.

[Example 9-4](#) provides an example of a join condition.

Example 9-4 Join Condition in Target Activity

```

<flow>
    <links>
        <link name="linkStatus2" />
    </links>
    <empty name="E2">
        <sources>
            <source linkName="linkStatus2">
                <transitionCondition>>false()</transitionCondition>
            </source>
        </sources>
    </empty>
    <empty name="E2">
        <targets>
            <joinCondition>bpws:getLinkStatus('linkStatus2')=true()</joinCondition>
            <target linkName="linkStatus2" />
        </targets>
    </empty>
</flow>

```

9.3 Customizing the Number of Parallel Branches

This section describes how to customize the number of parallel branches with the following activities:

- A FlowN activity in a BPEL version 1.1 project
- A forEach activity in a BPEL version 2.0 project

9.3.1 Customizing the Number of Flow Activities with the flowN Activity in BPEL 1.1

In the flow activity, the BPEL code determines the number of parallel branches. However, often the number of branches required is different depending on the available information. The flowN activity creates multiple flows equal to the value of N, which is defined at runtime based on the data available and logic within the

process. An index variable increments each time a new branch is created, until the index variable reaches the value of *N*.

The *flowN* activity performs activities on an arbitrary number of data elements. As the number of elements changes, the BPEL process service component adjusts accordingly.

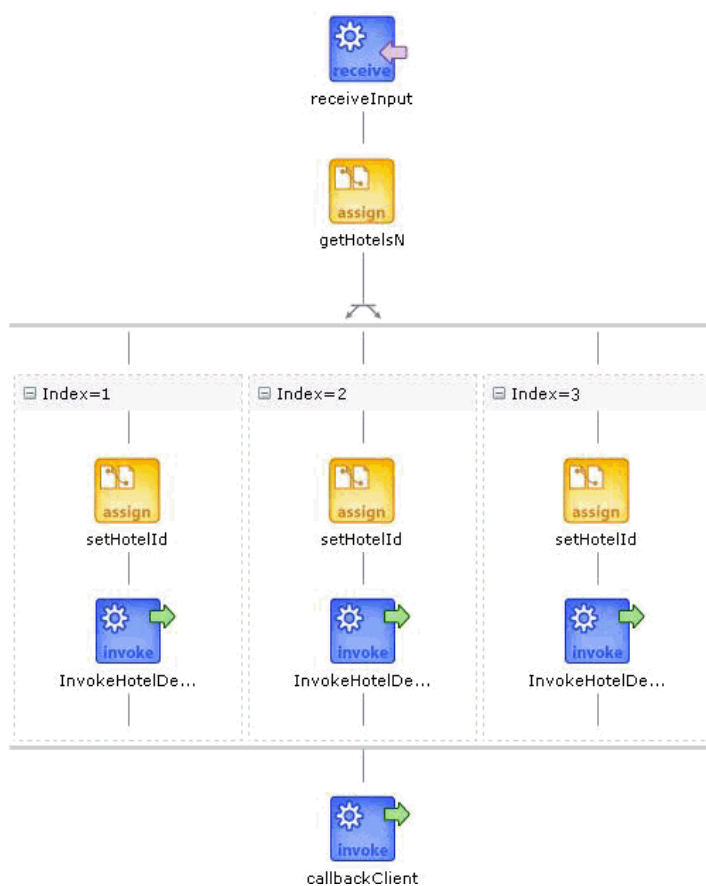
The branches created by *flowN* perform the same activities, but use different data. Each branch uses the index variable to look up input variables. The index variable can be used in the XPath expression to acquire the data specific for that branch.

For example, suppose there is an array of data. The BPEL process service component uses a *count* function to determine the number of elements in the array. The process then 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 9–9](#) shows the runtime flow of a *flowN* activity in Oracle Enterprise Manager Fusion Middleware Control 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 9–9 Oracle Enterprise Manager Fusion Middleware Control View of the Execution of a flowN activity



9.3.1.1 How to Create a flowN Activity

To create a flowN activity:

1. In the Component Palette, expand **Oracle Extensions**.
2. Drag a **FlowN** activity into the designer.
3. Click the + sign to expand the **FlowN** activity.
4. Double-click the **FlowN** activity.

Figure 9–10 shows the flowN dialog.

Figure 9–10 FlowN Dialog

The FlowN dialog box has a title bar with a close button. It contains five tabs: General, Annotations, Skip Condition, Targets, and Sources. The General tab is active. It has three input fields: 'Name:' with the value 'FlowNCR2', 'N:' with the value '2', and 'Index Variable:' with the value 'FlowNCR2_Variable'. There are icons for a calculator and a search function next to the 'N' and 'Index Variable' fields respectively. At the bottom, there are four buttons: Help, Apply, OK, and Cancel.

The flowN dialog enables you to:

- Name the activity
 - Enter a value or an expression for calculating the value of N (the number of branches to create)
 - Define the index variable (the time to wait in each branch)
5. Drag and define additional activities in the flowN activity.

Figure 9–11 shows how a FlowN activity appears with additional activities.

Figure 9–11 FlowN Activity with Additional Activities

9.3.1.2 What Happens When You Create a FlowN Activity

The following code shows the .bpel file that uses the flowN activity to look up information on an arbitrary number of hotels. The following actions take place.

Example 9–5 shows the sequence name.

Example 9–5 Sequence Name

```

<sequence name="main">
  <!-- Received input from requester.
    Note: This maps to operation defined in NflowHotels.wsdl

```

The requester sends a set of hotels names wrapped into the "inputVariable"
-->

A receive activity calls the client partner link to get the information that the flowN activity must define N times and look up the hotel information. [Example 9-6](#) provides an example.

Example 9-6 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
    nodes passed in.
    An intermediate variable called "NbParallelFlow" is
    used to store the number of N flows being executed
-->
<assign name="getHotelsN">
  <copy>
    <from
expression="count($InputVariable.payload/client:HotelName);"/>
    <to variable="NbParallelFlow"/>
  </copy>
</assign>
<!-- Initiating the FlowN activity
    The N value is initialized with the value stored in the
    "NbParallelFlow" variable
    The variable call "Index" is defined as the index variable
    NOTE: Both "NbParallelFlow" and "Index" variables have to be declared
-->
```

The flowN activity begins next. After defining a name for the activity of flowN, N is defined as a value from the inputVariable, which is the number of hotel entries. The activity also assigns index as the index variable. [Example 9-7](#) provides an example.

Example 9-7 FlowN Activity

```
<bpelx:flowN name="FlowN" N="bpws:getVariableData('NbParallelFlow')
indexVariable="Index">
  <sequence name="Sequence_1">
    <!-- Fetching each hotelName by indexing the "inputVariable" with the
    "Index" variable.
        Note the usage of the "concat()" Xpath function to create the
    expression accessing the array element.
    -->
```

The copy rule shown in [Example 9-8](#) then uses the index variable to concatenate the hotel entries into a list:

Example 9-8 Assign Activity

```
<assign name="setHotelId">
  <copy>
    <from expression=
"bpws:getVariableData('inputVariable','payload',concat('/client:Nflo
wHotelsProcessRequest/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 9-9](#) provides an example.

Example 9-9 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 does not do anything with the retrieved information.
In real life, it could then be used to continue the process.
Note: Meanwhile an indexing variable is used. Unlike a while loop, the
activities are executed in parallel, not sequentially.
-->
</sequence>
</bpelx:flowN>

```

Finally, the BPEL process sends detailed information on each hotel to the client partner link. [Example 9-10](#) provides an example.

Example 9-10 Invoke Activity

```

<invoke name="callbackClient" partnerLink="client"
portType="client:NflowHotelsCallback" operation="onResult"
inputVariable="outputVariable"/>
</sequence>
</sequence>

```

9.3.2 Processing Multiple Sets of Activities with the `forEach` Activity in BPEL 2.0

You can use a `forEach` activity to process multiple sets of activities sequentially or in parallel. The `forEach` activity executes a contained (child) scope activity exactly $N+1$ times, where N equals a final counter value minus a starting counter value that you specify in the **Counter Values** tab of the For Each dialog. While other structured activities such as a flow activity can have any type of activity as its contained activity, the `forEach` activity can only include a scope activity.

When the `forEach` activity is started, the expressions you specify for the starting counter and final counter values are evaluated. Once the two values are returned, they remain constant for the lifecycle of the activity. Both expressions must return a value containing at least one character. If these expressions do not return valid values, a fault is thrown. If the starting counter value is greater than the final counter value, the contained scope activity is not performed and the `forEach` activity is considered complete.

During each iteration, the variable specified in the **Counter Name** field on the **General** tab is implicitly declared in the `forEach` activity's contained scope. During the first iteration of the scope, the counter variable is initialized with the starting counter value. The next iteration causes the counter variable to be initialized with the starting counter value, plus one. Each subsequent iteration increments the previously initialized counter variable value by one until the final iteration, where the counter is set to the final counter value. The counter variable is local to the enclosed scope activity. Although its value can be changed during an iteration, that value is lost after each

iteration. Therefore, the counter variable value does not impact the value of the next iteration's counter.

The `forEach` activity supports the following looping iterations:

- Sequential (default)

The `forEach` activity performs looping iterations sequentially N times over a given set of activities defined within a scope activity. As an example, the `forEach` activity iterates over an incoming purchase order message where the purchase order message consists of N order items. The enclosed scope activity must be executed $N+1$ times, with each instance starting only after the previous iteration has completed.

- Parallel

All looping iterations are started at the same time and processed in parallel. Parallel iterations are useful in environments in which sets of independent data are processed or independent interaction with different partners is performed in parallel. To enable parallel looping, you select the **Parallel Execution** checkbox on the **General** tab. In these scenarios, execution of the $N+1$ instances of the contained scope activity occurs in parallel. Each copy of the scope activity has the same counter variable that you specify in the **Counter Name** field of the **General** tab declared in the same way as specified for a sequential `forEach` activity. Each instance's counter variable must be uniquely initialized in parallel with one of the integer values beginning with the starting counter value and proceeding up to and including the final counter value.

Unlike a flow activity, the number of parallel branches is not known at design time with the `forEach` activity. The specified counter variable iterates through the number of parallel branches, controlled by the starting counter value and final counter value.

You can also specify a completion condition on the **Completion** tab. This condition enables the `forEach` activity to execute the condition and complete without executing or finishing all the branches specified. As an example, you send out parallel requests and a sufficient subset of the recipients have responded. A completion condition is optionally specified to prevent the following:

- Some children from executing (in the sequential case)
- To force early termination of some of the children (in the parallel case)

If you do not specify a completion condition, the `forEach` activity completes when the contained scope has completed.

If a premature termination occurs (due to a fault or the completion condition evaluating to `true`), then the $N+1$ requirement does not apply.

[Example 9-11](#) shows the `forEach` activity syntax.

Example 9-11 `forEach` Activity

```
<forEach counterName="MyVariableName" parallel="yes|no"
  standard-attributes>
  standard-elements
  <startCounterValue expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </startCounterValue>
  <finalCounterValue expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </finalCounterValue>
```

```

<completionCondition>?
  <branches expressionLanguage="anyURI"?
    successfulBranchesOnly="yes|no"?>?
    unsigned-integer-expression
  </branches>
</completionCondition>
<scope ..>...</scope>
</forEach>

```

Note: The `successfulBranchesOnly` attribute is not supported for this release.

9.3.2.1 How to Create a `forEach` Activity

To create a `forEach` activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag a **For Each** activity into the designer, as shown in [Figure 9–12](#).

Note the contained scope activity in the `forEach` activity.

Figure 9–12 *Contained Scope Activity in a `forEach` Activity*



3. Double-click the **ForEach** activity.
4. In the **Counter Name** field of the **General** tab, enter a counter value name, as shown in [Figure 9–13](#).

Note the **Parallel Execution** checkbox. If this checkbox is selected, all looping iterations are started at the same time and processed in parallel.

Figure 9–13 General Tab of the forEach Activity

For Each

Completion Documentation Targets Sources

General Counter Values

Name:

Counter Name:

☐ Parallel Execution

Help Apply OK Cancel

5. Click the **Counter Values** tab.
6. Enter the starting counter value and final counter value, as shown in [Figure 9–14](#).

Figure 9–14 Counter Values Tab of the forEach Activity

For Each

Completion Documentation Targets Sources

General Counter Values

Start Value

Expression Language: XPath 1.0 in BPEL 2.0

Expression:

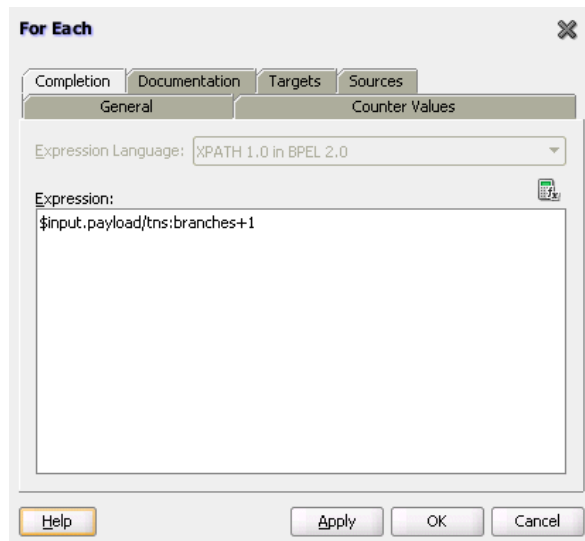
Final Value

Expression Language: XPath 1.0 in BPEL 2.0

Expression:

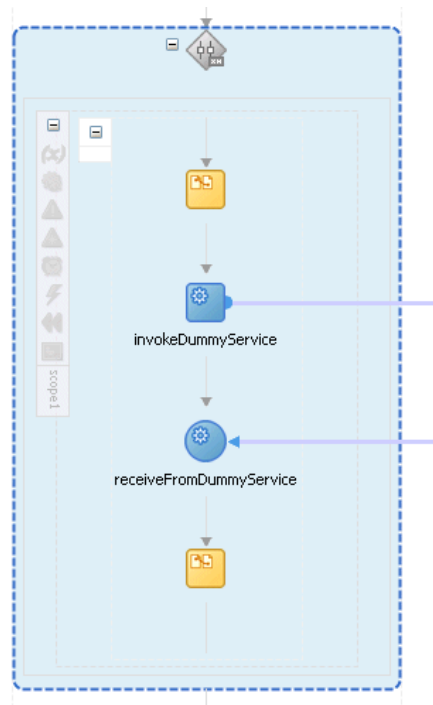
Help Apply OK Cancel

7. Click the **Completion** tab.
8. If you want to specify a completion condition that enables the forEach activity to execute the condition and complete without executing or finishing all the branches specified, click the **XPath Expression Builder** icon in the **Expression** field to enter a condition. [Figure 9–15](#) provides details.

Figure 9–15 Completion Tab of the forEach Activity

9. Click **Apply**, then **OK**.
10. Expand the contained **Scope** activity of the **ForEach** activity.
11. Design the enclosed **Scope** activity.

When complete, the **forEach** and contained **scope** activity can appear similar in structure to that shown in [Figure 9–16](#).

Figure 9–16 forEach Activity with Contained and Expanded Scope Activity

9.3.2.2 What Happens When You Create a forEach Activity

[Example 9–12](#) shows the .bpe1 file after design is complete for a sequential **forEach** activity.

Example 9-12 *forEach Activity - Sequential*

```

<faultHandlers>
  <catch faultName="bpel:invalidBranchCondition">
<sequence>
  <assign>
    <copy>
      <from>'invalidBranchCondition happened'</from>
      <to>$output.payload</to>
    </copy>
  </assign>

  <reply name="replyOutput" partnerLink="client"
    portType="tns:Test" operation="process" variable="output"/>
</sequence>
  </catch>
</faultHandlers>
<sequence>
  <!-- pick input from requestor -->
  <receive name="receive" createInstance="yes"
    partnerLink="client" portType="tns:Test"
    operation="process" variable="input"/>
  <assign>
    <copy>
      <from>3</from>
      <to>$request.payload</to>
    </copy>
    <copy>
      <from>' '</from>
      <to>$output.payload</to>
    </copy>
  </assign>

  <forEach counterName="i" parallel="no">
    <startCounterValue>$input.payload/tns:startCounter+1</startCounterValue>
    <finalCounterValue>$input.payload/tns:finalCounter+1</finalCounterValue>
    <completionCondition>
      <branches>$input.payload/tns:branches+1</branches>
    </completionCondition>
    <scope name="scope1">
      <partnerLinks>
        <partnerLink name="DummyService" partnerLinkType="tns:DummyService"
          myRole="DummyServiceClient" partnerRole="DummyServiceProvider"/>
      </partnerLinks>
      <sequence>
        <assign>
          <copy>
            <from>concat($output.payload, $i, 'A')</from>
            <to>$output.payload</to>
          </copy>
        </assign>
        <invoke name="invokeDummyService" partnerLink="DummyService"
          portType="tns:DummyPortType"
          operation="initiate" inputVariable="request"/>
        <receive name="receiveFromDummyService" partnerLink="DummyService"
          portType="tns:DummyCallbackPortType"
          operation="onResult" variable="response"/>
        <assign>
          <copy>
            <from>concat($output.payload, $i, 'B')</from>
            <to>$output.payload</to>

```

```

        </copy>
      </assign>
    </sequence>
  </scope>
</forEach>

<!-- respond output to requestor -->
<reply name="replyOutput" partnerLink="client"
      portType="tns:Test" operation="process" variable="output"/>
</sequence>

```

[Example 9–13](#) shows the .bpel file after design is complete for a parallel `forEach` activity.

Example 9–13 *forEach Activity - Parallel*

```

<sequence>
  <!-- pick input from requestor -->
  <receive name="receive" createInstance="yes"
        partnerLink="client" portType="tns:Test"
        operation="process" variable="input"/>
  <assign>
    <copy>
      <from>${input.payload/tns:value1}</from>
      <to>${request.payload}</to>
    </copy>
    <copy>
      <from>' '</from>
      <to>${output.payload}</to>
    </copy>
  </assign>
  <forEach counterName="i" parallel="yes">
    <startCounterValue>(${input.payload/tns:value1} + 1)</startCounterValue>
    <finalCounterValue>(${input.payload/tns:value2} + 2)</finalCounterValue>
    <scope name="scope1">
      <partnerLinks>
        <partnerLink name="DummyService" partnerLinkType="tns:DummyService"
              myRole="DummyServiceClient" partnerRole="DummyServiceProvider"/>
      </partnerLinks>
      <sequence>
        <assign>
          <copy>
            <from>concat(${output.payload}, 'A')</from>
            <to>${output.payload}</to>
          </copy>
        </assign>
        <invoke name="invokeDummyService" partnerLink="DummyService"
              portType="tns:DummyPortType"
              operation="initiate" inputVariable="request"/>
        <receive name="receiveFromDummyService" partnerLink="DummyService"
              portType="tns:DummyCallbackPortType"
              operation="onResult" variable="response"/>
        <assign>
          <copy>
            <from>concat(${output.payload}, 'B')</from>
            <to>${output.payload}</to>
          </copy>
        </assign>
      </sequence>
    </scope>
  </forEach>

```

```
</forEach>
<!-- respond output to requestor -->
<reply name="replyOutput" partnerLink="client"
      portType="tns:Test" operation="process" variable="output" />
</sequence>
```

Using Conditional Branching in a BPEL Process

This chapter describes how to use conditional branching in a BPEL process service component. Conditional branching introduces decision points to control the flow of execution of a BPEL process service component.

This chapter includes the following sections:

- [Section 10.1, "Introduction to Conditional Branching"](#)
- [Section 10.2, "Defining Conditional Branching"](#)
- [Section 10.3, "Creating a While Activity to Define Conditional Branching"](#)
- [Section 10.4, "Creating a repeatUntil Activity to Define Conditional Branching"](#)
- [Section 10.5, "Specifying XPath Expressions to Bypass Activity Execution"](#)

For additional information on creating conditional branching in a SOA composite application, see the Fusion Order Demo application.

10.1 Introduction to Conditional Branching

BPEL applies logic to make choices through conditional branching. You can use the following activities to design your code to select different actions based on conditional branching:

- Switch activity (in a BPEL version 1.1 project)

Enables you to set up two or more branches, with each branch in the form of an XPath expression. If the expression is true, then the branch is executed. If the expression is false, then the BPEL process service component moves to the next branch condition, until it either finds a valid branch condition, encounters an otherwise branch, or runs out of branches. If multiple branch conditions are true, then BPEL executes the first true branch. For information about how to create switch activities, see [Section 10.2.1, "Defining Conditional Branching with the Switch Activity in BPEL 1.1."](#)
- If activity (in a BPEL version 2.0 project)

Enables you to use an if activity when conditional behavior is required for specific activities to decide between two or more branches. The if activity replaces the switch activity that appeared in BPEL 1.1 processes. For information about how to create if activities, see [Section 10.2.2, "Defining Conditional Branching with the If Activity in BPEL 2.0."](#)
- While activity

Enables you to create a while loop to select between two actions. [Section 10.3, "Creating a While Activity to Define Conditional Branching"](#) describes while activities.

Many branches are set up, and each branch has a condition in the form of an XPath expression.

You can program a conditional branch to have a timeout. That is, if a response cannot be generated in a specified period, the BPEL flow can stop waiting and resume its activities. [Chapter 14, "Using Events and Timeouts in BPEL Processes"](#) explains this feature in detail.

Note: You can also define conditional branching logic with business rules. See *Oracle Fusion Middleware User's Guide for Oracle Business Rules* and the WebLogic Fusion Order Demo application for details.

10.2 Defining Conditional Branching

This section describes how to define conditional branching with the following activities:

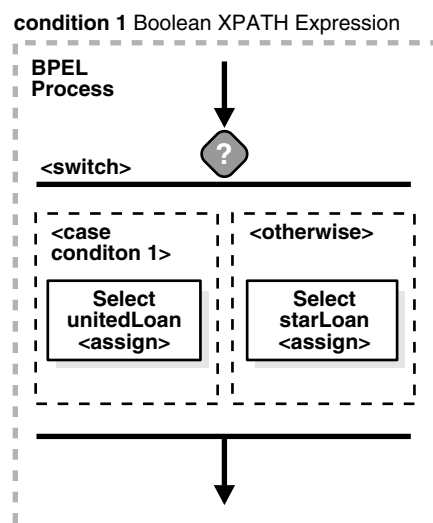
- Switch activity in a BPEL version 1.1 project
- If activity in a BPEL version 2.0 project

10.2.1 Defining Conditional Branching with the Switch Activity in BPEL 1.1

Assume you designed a flow activity in the BPEL process service component that gathered loan offers from two companies at the same time, but did not compare either of the offers. Each offer was stored in its own global variable. To compare the two bids and make decisions based on that comparison, you can use a switch activity.

[Figure 10–1](#) provides an overview of a BPEL conditional branching process that has been defined in a switch activity.

Figure 10–1 Conditional Branching



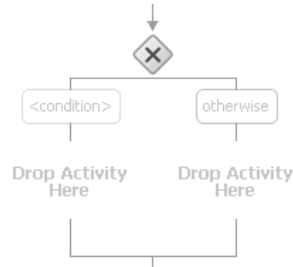
10.2.1.1 How to Create a Switch Activity

To create a switch activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag a **Switch** activity into the designer, as shown in [Figure 10–2](#).

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 10–2 Switch Activity



3. In the first branch, double-click the **condition** box.

A dialog for entering a condition is displayed, as shown in [Figure 10–3](#).

Figure 10–3 Condition Dialog

4. In the **Label** field, enter a name for the condition branch. When complete, this name is displayed in Oracle BPEL Designer.
5. In the **Description** field, enter a description of the capabilities of this condition branch.
6. In the **Condition** field, click the **Expression Builder** icon to access the Expression Builder dialog.
7. Create your expression.

```
bpws:getVariableData('loanOffer1','payload','/loanOffer/APR') >
bpws:getVariableData('loanOffer2','payload','/loanOffer/APR')
```

In this example, two loan offers from completing loan companies are stored in the global variables `loanOffer1` and `loanOffer2`. Each loan offer variable contains the loan offer's APR. The BPEL flow must choose the loan with the lower APR.

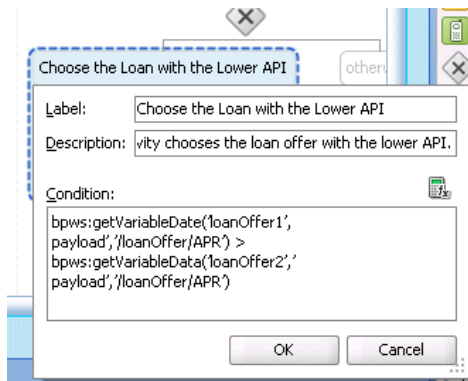
One of the following switch activities takes place:

- If `loanOffer1` has the higher APR, then the first branch selects `loanOffer2` by assigning the `loanOffer2` payload to the selectedLoanOffer payload.

- If `loanOffer1` does *not* have the lower APR than `loanOffer2`, the otherwise case assigns the `loanOffer1` payload to the `selectedLoanOffer` payload.
8. Click **OK**.

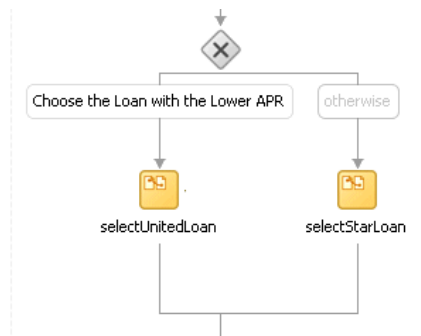
The expression is displayed. The value you entered in the **Label** field of the dialog becomes the name of the condition branch. [Figure 10-4](#) provides details.

Figure 10-4 Completed Condition Dialog



9. Click **OK**.
10. Add and configure additional activities as needed. [Figure 10-5](#) provides details.

Figure 10-5 Switch Activity Design



10.2.1.2 What Happens When You Create a Switch Activity

A switch activity, like a flow activity, has multiple branches. In [Example 10-1](#), there are only two branches shown in the `.bpel` file after design completion. The first branch, which selects a loan offer from a company named United Loan, is executed if a case condition containing an XPath boolean expression is met. Otherwise, the second branch, which selects the offer from a company named Star Loan, is executed. By default, the switch activity provides two switch cases, but you can add more if you want.

Example 10-1 Switch Activity

```
<switch name="switch-1">
  <case condition="bpws:getVariableData('loanOffer1', 'payload',
    '/autoloan:loanOffer/autoloan:APR') >
    bpws:getVariableData('loanOffer2', 'payload', '/autoloan:loanOffer/autoloan:APR
  ')">
```

```

" name="Choose_the_Loan_with_the_Lower_APR">
  <bpelx:annotation>
    <bpelx:general>
      <bpelx:property name="userLabel">Choose the Loan with
        the Lower APR</bpelx:property>
    </bpelx:general>
  </bpelx:annotation>
  <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>

```

10.2.2 Defining Conditional Branching with the If Activity in BPEL 2.0

You can use an if activity when conditional behavior is required for specific activities to decide between two or more branches. Only one activity is selected for execution from a set of branches. The if activity consists of a list of one or more conditional branches that are considered for execution in the following order:

- The if branch
- Optional elseif branches
- An optional else branch

The first branch whose condition evaluates to true is taken, and its contained activity is performed. If no branch with a condition is taken, then the else branch is taken (if present). The if activity is complete when the contained activity of the selected branch completes, or immediately when no condition evaluates to true and no else branch is specified.

The if activity is a BPEL version 2.0 feature that replaces the switch activity that was included in BPEL version 1.1.

[Example 10–2](#) shows the if activity syntax.

Example 10–2 If Activity

```

<if standard-attributes>
  standard-elements
  <condition>some conditon expression</condition>
  activity
  <elseif>*
    <condition>some condition expression</condition>
    some activity
  </elseif>
  <else>?
    some activity

```

```

    </else>
  </if>

```

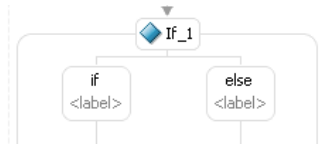
10.2.2.1 How to Create an If Activity

To create an If activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag an **If** activity into the designer.

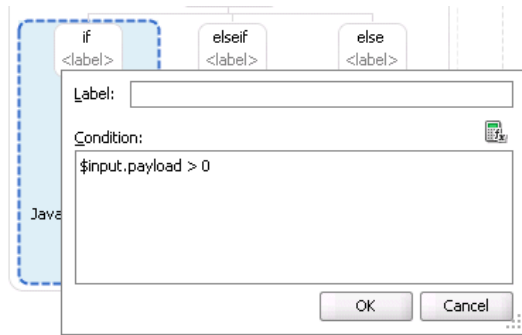
The **if** and **else** conditions are displayed, as shown in [Figure 10–6](#).

Figure 10–6 If Activity



3. If you want to add **elseif** conditions, highlight the **If** activity, and select the **Add** icon to invoke a menu.
4. Click the **if** branch.
5. In the **Condition** field, enter a condition, as shown in [Figure 10–7](#). You can also click the **XPath Expression Builder** icon to invoke the Expression Builder dialog.

Figure 10–7 if Branch of the If Activity



6. Click **OK**.
7. Drag and define additional activities into the **if** condition, as needed. These activities are executed if the if condition evaluates to true.
8. Click the **elseif** branch (if you added this branch).
9. In the **Condition** field, enter a condition, as shown in [Figure 10–8](#).

Figure 10–8 elseif Branch of the If Activity

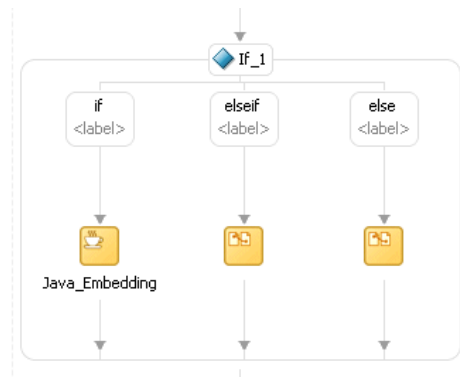
10. Click **OK**.

11. Drag and define additional activities into the **elseif** condition, as needed. These activities are executed if the if branch did not evaluate to true, and this elseif branch evaluates to true.

12. Click the **else** label.

13. Enter a condition or drag and define additional activities into the **else** condition, as needed. These activities are executed if the if and any elseif branches did not evaluate to true, and this else branch evaluates to true.

Figure 10–9 shows a completed if activity in which each branch includes contained activities.

Figure 10–9 Completed If Activity

10.2.2.2 What Happens When You Create an If Activity

Example 10–3 provides an example of the `.bpel` file after design completion. The if activity has if, elseif, and else branches defined. The first branch to evaluate to true is executed.

Example 10–3 If Activity

```
<sequence>
  <!-- receive input from requestor -->
  <receive name="receiveInput" partnerLink="client" portType="tns:Test"
    operation="process" variable="input" createInstance="yes"/>
  <!-- assign default value -->
  <assign>
    <copy>
```

```

        <from>'Value is greater than zero'</from>
        <to>$output.payload</to>
    </copy>
</assign>
<copy>
    <from>'Value is greater than zero'</from>
    <to>$output.payload</to>
</copy>
</assign>
<!-- switch depends on the input value field -->
<if>
    <condition>$input.payload > 0</condition>
    <extensionActivity>
        <bpelx:exec name="Java_Embedding" version="1.5" language="java">
            System.out.println("if condition is true.\n");
        </bpelx:exec>
    </extensionActivity>
    <elseif>
        <condition>bpws:getVariableData('input', 'payload') &lt; 0</condition>
        <assign>
            <copy>
                <from>'Value is less than zero'</from>
                <to>$output.payload</to>
            </copy>
        </assign>
    </elseif>
    <else>
        <assign>
            <copy>
                <from>'Value is equal to zero'</from>
                <to>$output.payload</to>
            </copy>
        </assign>
    </else>
</if>

<!-- respond output to requestor -->
<reply name="replyOutput" partnerLink="client"
    portType="tns:Test" operation="process" variable="output"/>
</sequence>

```

10.3 Creating a While Activity to Define Conditional Branching

Another way to design your BPEL code to select between multiple actions is to use a while activity to create a while loop. The while loop repeats an activity until a specified success criteria is met. For example, if a critical web service is returning a service busy message in response to requests, you can use the while activity to keep polling the service until it becomes available. The condition for the while activity is that the latest message received from the service is busy, and the operation within the while activity is to check the service again. Once the web service returns a message other than service busy, the while activity terminates and the BPEL process service component continues, ideally with a valid response from the web service.

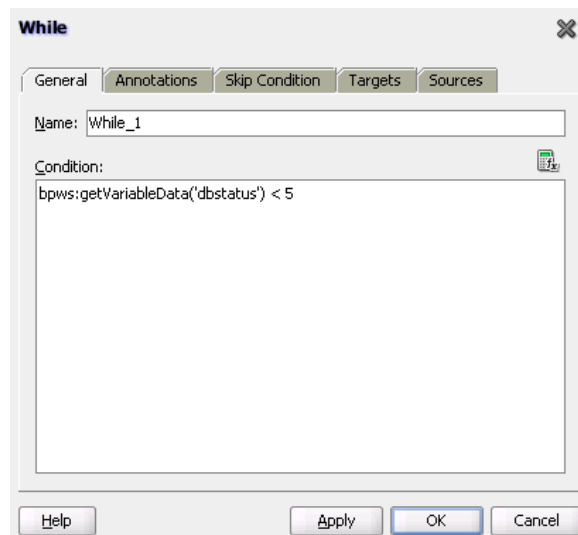
10.3.1 How To Create a While Activity

To create a while activity:

1. In the Component Palette, expand **BPEL Constructs**.

2. Drag a **While** activity into the designer.
3. Click the + sign to expand the while activity.
The while activity has icons to allow you to build condition expressions and to validate the while definition. It also provides an area for you to drag an activity to define the while loop.
4. Drag and define additional activities for using the while condition into the **Drop Activity Here** area of the **While** activity (for example, a **Scope** activity).
The activities can be existing or new activities.
5. Press Ctrl+Space to invoke the XPath Building Assistant or click the **XPath Expression Builder** icon to open the Expression Builder dialog.
6. Enter an expression to perform repeatedly, as shown in [Figure 10–10](#). This action is performed until the given boolean while condition is no longer true. In this example, this activity is set to loop while less than 5.

Figure 10–10 While Activity with an Expression



7. Click **OK** when complete.

10.3.2 What Happens When You Create a While Activity

[Example 10–4](#) provides an example of the .bpel file after design completion. The while activity includes a scope activity. The scope activity includes invoke, assign, and wait activities. Database exception handling tasks are performed by creating a local variable and placing the invoke activity inside the scope activity. The local variable is set to false (represented by 0). You attempt to call the external partner service in the while loop activity until the local variable is satisfied (set to 1). The while activity is set to loop a maximum of five times. In the case of an exception, you reset the flag to false (0).

Example 10–4 While Activity

```
<while name="While_1" condition="bpws:getVariableData('dbStatus') > 5">
  <scope name="Scope_1">
<faultHandlers>
  <catchAll>
```

```

        <sequence name="Sequence_2">
            <assign name="assign_DB_retry">
                <copy>
                    <from expression="bpws:getVariableData('dbStatus') + 1"/>
                    <to variable="dbStatus"/>
                </copy>
            </assign>
            <wait name="Wait_30_sec" for="'PT31S'"/>
        </sequence>
    </catchAll>
</faultHandlers>
<sequence name="Sequence_1">
    <invoke name="Write_DBWrite" partnerLink="WriteDBRecord"
        portType="ns2:WriteDBRecord_ptt" operation="insert"
        inputVariable="Invoke_DBWrite_merge_InputVariable"/>
    <assign name="Assign_dbComplete">
        <copy>
            <from expression="'10'"/>
            <to variable="dbStatus"/>
        </copy>
    </assign>
</sequence>
</scope>
</while>

```

10.4 Creating a repeatUntil Activity to Define Conditional Branching

If the body of an activity must be performed at least once, use a repeatUntil activity instead of a while activity. The XPath expression condition in the repeatUntil activity is evaluated after the body of the activity completes. The condition is evaluated repeatedly (and the body of the activity processed) until the provided boolean condition is true.

Note: This activity is supported in BPEL version 2.0 projects.

10.4.1 How to Create a repeatUntil Activity

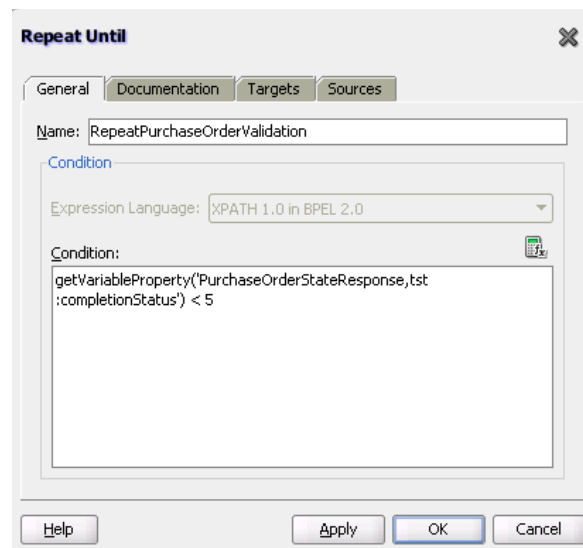
To create a repeatUntil activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag a **Repeat Until** activity into the designer.
3. Double-click the **Repeat Until** activity.
4. Enter a name or accept the default value.
5. In the **Condition** field, click the **XPath Expression Builder** icon to enter an XPath expression condition.

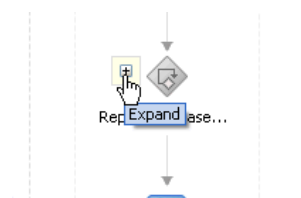
The Expression Builder dialog is displayed.

6. Enter a boolean XPath expression condition, and click **OK**.

The condition you entered is displayed in the Repeat Until dialog, as shown in [Figure 10–11](#).

Figure 10–11 Completed Repeat Until Dialog

7. Click **Apply**, then **OK**.
8. Expand the **Repeat Until** activity, as shown in [Figure 10–12](#).

Figure 10–12 repeatUntil Activity Being Expanded

9. Design the body of the activity by dragging in activities from the Component Palette and defining their property values. These activities are evaluated until the XPath expression condition is evaluated to true.

10.4.2 What Happens When You Create a repeatUntil Activity

[Example 10–5](#) provides an example of the .bpel file after design completion. In this scenario, purchase order validation must be performed at least once, then repeatedly, based on evaluating the completion status until the status is updated to 5.

Example 10–5 repeatUntil Activity

```
<repeatUntil>
  <sequence>
    <invoke name="PurchaseOrderValidation" ... />
    <receive name="receiveValidation"
      partnerLink="PurchaseOrderValidation"
      operation="returnPurchaseOrderValidation"
      variable="PurchaseOrderStatusResponse" />
  </sequence>
  <condition>
    bpel:getVariableProperty(
      "PurchaseOrderStatusResponse", "tst:completionStatus") < 5
  </condition>
</repeatUntil>
```

10.5 Specifying XPath Expressions to Bypass Activity Execution

You can specify an XPath expression in an activity that, when evaluated to true, causes that activity to be skipped. This functionality provides an alternative to using a switch activity for conditionally executing activities. The skip condition for activities is specified as follows:

```
<activity bpelx:skipCondition="boolean-expr"/>
```

The `bpelx:skipCondition` attribute causes an XPath expression to be evaluated immediately upon creation of the activity instance. If the skip expression returns a false boolean value, the activity is executed. If the skip expression returns a true boolean value, the activity is completed immediately and execution moves to the activity immediately following that one.

This construct is equivalent to a switch/case structured activity with a single case element with a condition that is the opposite of the skip condition.

Note: The skip condition is only available in BPEL projects that support BPEL version 1.1.

[Example 10–6](#) provides an example of `bpelx:skipCondition` attribute use. If `myvalue` is 0, the expression evaluates to true, and the assign activity is skipped. If `myvalue` is 10, the expression evaluates to false, and the copy operation of the assign activity is executed.

Example 10–6 Use of `bpelx:skipCondition` Attribute

```
<assign bpelx:skipCondition="bpws:getVariableData('input',
  'payload','/tns:inputMsg/tns:myvalue') <= 0">
  <copy>
    <from expression="'Value is greater than zero'"/>
    <to variable="output" part="payload"
  query="/tns:resultMsg/tns:valueResult"/>
  </copy>
</assign>
```

The equivalent functionality used with a switch activity is shown in [Example 10–7](#).

Example 10–7 Equivalent Functionality with a Switch Activity

```
<switch>
  <case condition="bpws:getVariableData('input',
    'payload','/tns:inputMsg/tns:value') > 0">
    <assign>
      <copy>
        <from expression="'Value is greater than zero'"/>
        <to variable="output" part="payload"
      query="/tns:resultMsg/tns:valueResult"/>
      </copy>
    </assign>
  </case>
</switch>
```

You can also use built-in and custom XPath functions and `$variable` references within the skip condition expression. [Example 10–8](#) provides several examples:

Example 10–8 Built-in and Custom XPath Functions and \$variable References

```
<assign bpelx:skipCondition="bpws:getVariableData( 'crOutput', 'payload',
  '/tns:rating' ) > 0">

<assign bpelx:skipCondition="custom:validateRating()" ... />

<assign xmlns:fn='http://www.w3.org/2005/xpath-functions'
  bpelx:skipCondition="fn:false()" ... />
```

If an error is thrown by the XPath expression evaluation, the error is wrapped with a BPEL fault and thrown from the activity.

An event is added to the BPEL instance audit trail for activities that are bypassed due to the skip condition expression evaluating to true. Even if the skip condition evaluates to false (meaning the activity is performed), the fact that a skip condition expression was evaluated is still logged to the audit trail for debugging purposes.

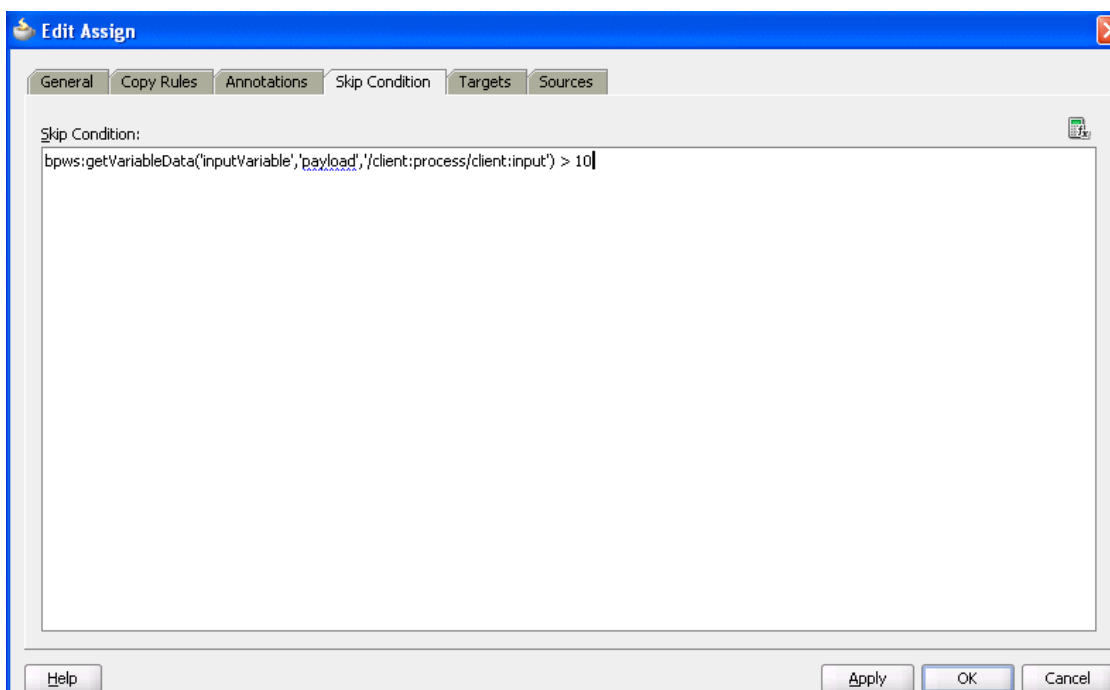
If the XPath engine fails to evaluate the boolean value, `bpws:subLanguageFault` is thrown. This is the same fault thrown when a switch/case condition does not evaluate to a boolean value. This is also logged to the audit trail for debugging purposes.

10.5.1 How to Specify XPath Expressions to Bypass Activity Execution

To specify XPath expressions to bypass activity execution:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag the activity into the designer in which to create the skip condition.
3. Click the **Skip Condition** tab.
4. Specify an XPath expression that, when evaluated to true, causes an activity to be skipped. [Figure 10–13](#) provides details.

Figure 10–13 Skip Condition XPath Expression



5. Click **Apply**, then **OK**.

10.5.2 What Happens When You Specify XPath Expressions to Bypass Activity Execution

The code segment in the `.bpel` file defines the specific operation after design completion.

For example, the XPath expression shown in [Example 10–9](#), when evaluated to true (for example, input is 20), causes the assign activity to be skipped.

Example 10–9 *skipCondition Attribute For Bypassing Activity Execution*

```
<sequence name="main">
  . . .
  . . .
  <assign name="Assign_1"

    bpelx:skipCondition="number(bpws:getVariableData('inputVariable','payload','client:
process/client:input')) > 10">
    <copy>
      <from expression="'Assign Block is not Skipped'"/>
      <to variable="inputVariable" part="payload"
        query="/client:process/client:input"/>
    </copy>
  </assign>
  . . .
  . . .
</sequence>
```

Using Fault Handling in a BPEL Process

This chapter describes how to use fault handling in a BPEL process. Fault handling allows a BPEL process service component to handle error messages or other exceptions returned by outside web services, and to generate error messages in response to business or runtime faults. You can also define a fault management framework to catch faults and perform user-specified actions defined in a fault policy file.

This chapter includes the following sections:

- [Section 11.1, "Introduction to a Fault Handler"](#)
- [Section 11.2, "Introduction to BPEL Standard Faults"](#)
- [Section 11.3, "Introduction to Categories of BPEL Faults"](#)
- [Section 11.4, "Using the Fault Management Framework"](#)
- [Section 11.5, "Catching BPEL Runtime Faults"](#)
- [Section 11.6, "Getting Fault Details with the getFaultAsString XPath Extension Function"](#)
- [Section 11.7, "Throwing Internal Faults"](#)
- [Section 11.8, "Rethrowing Faults with the Rethrow Activity"](#)
- [Section 11.9, "Returning External Faults"](#)
- [Section 11.10, "Using a Scope Activity to Manage a Group of Activities"](#)
- [Section 11.11, "Re-executing Activities in a Scope Activity with the Replay Activity"](#)
- [Section 11.12, "Using Compensation After Undoing a Series of Operations"](#)
- [Section 11.13, "Stopping a Business Process Instance"](#)
- [Section 11.14, "Throwing Faults with Assertion Conditions"](#)

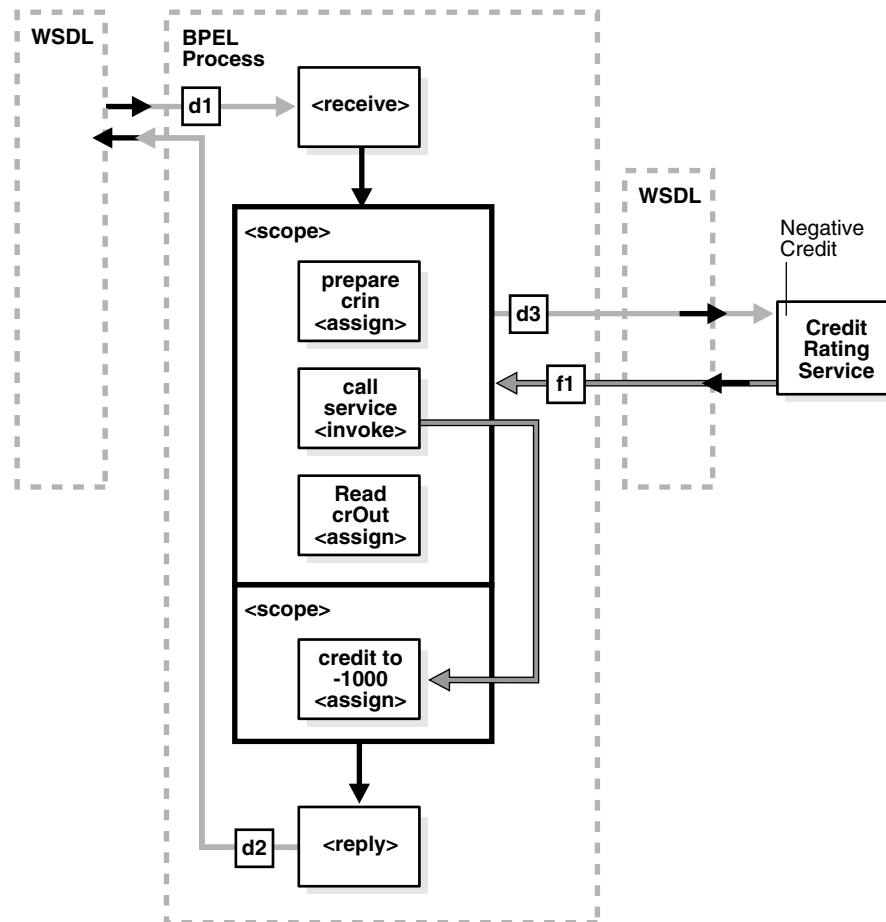
For additional information on creating fault handling in a SOA composite application, see the Fusion Order Demo application.

11.1 Introduction to a Fault Handler

Fault handlers define how the BPEL process service component responds when 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 11-1 provides an example of how a fault handler sets a credit rating variable to -1000.

Figure 11-1 Fault Handling



The code segment in [Example 11-1](#) defines the fault handler for this operation in the BPEL file:

Example 11-1 Fault Handler Definition

```

<faultHandlers>
  <catch faultName="services:NegativeCredit" faultVariable="crError">
    <assign name="crin">
      <copy>
        <from expression="-1000">
        </from>
        <to variable="input" part="payload"
          query="/autoloan:loanApplication/autoloan:creditRating"/>
      </copy>
    </assign>
  </catch>
</faultHandlers>

```

The `faultHandlers` tag contains the fault handling code. Within the fault handler is a catch activity, which defines the fault name and variable, and the copy instruction that sets the `creditRating` variable to -1000.

When you select web services for the BPEL process service component, determine the possible faults that may be returned and set up a fault handler for each one.

11.2 Introduction to BPEL Standard Faults

This section identifies the standard faults for BPEL 1.1 and BPEL 2.0.

11.2.1 BPEL 1.1 Standard Faults

This section identifies the standard faults for BPEL 1.1. Unless otherwise noted below, the *Business Process Execution Language for Web Services Specification* defines the following standard faults in the namespace of

`http://schemas.xmlsoap.org/ws/2003/03/business-process/:`

- `bindingFault` (BPEL extension fault defined in `http://schemas.oracle.com/bpel/extension`)
- `conflictingReceive`
- `conflictingRequest`
- `correlationViolation`
- `forcedTermination`
- `invalidReply`
- `joinFailure`
- `mismatchedAssignmentFailure`
- `remoteFault` (BPEL extension fault defined in `http://schemas.oracle.com/bpel/extension`)
- `repeatedCompensation`
- `selectionFailure`
- `uninitializedVariable`

Standard faults are defined as follows:

- Typeless, meaning they do not have associated `messageTypes`
- Not associated with any Web Services Description Language (WSDL) message
- Caught without a fault variable:

```
<catch faultName="bpws:selectionFailure">
```

11.2.2 BPEL 2.0 Standard Faults

The following list specifies the standard faults defined within the WS-BPEL specification. All standard fault names are qualified with the standard WS-BPEL namespace.

- `ambiguousReceive`
- `completionConditionFailure`
- `conflictingReceive`
- `conflictingRequest`
- `correlationViolation`
- `invalidBranchCondition`

- `invalidExpressionValue`
- `invalidVariables`
- `joinFailure`
- `mismatchedAssignmentFailure`
- `missingReply`
- `missingRequest`
- `scopeInitializationFailure`
- `selectionFailure`
- `subLanguageExecutionFault`
- `uninitializedPartnerRole`
- `uninitializedVariable`
- `unsupportedReference`
- `xsltInvalidSource`
- `xsltStylesheetNotFound`

11.2.2.1 Fault Handling Order of Precedence in BPEL 2.0

In BPEL 2.0, the order of precedence for catching faults thrown without associated data is as follows:

- If there is a catch activity with a matching `faultName` value that does not specify a `faultVariable` attribute, the fault is sent to the identified catch activity.
- Otherwise, if there is a catchAll activity, the fault is sent to the catchAll fault handler.
- Otherwise, the fault is processed by the default fault handler.

In BPEL 2.0, the order of precedence for catching faults thrown with associated data is as follows:

- If there is a catch activity with a matching `faultName` value that does not specify a `faultVariable` attribute, the fault is sent to the identified catch activity.
- If the fault data is a WSDL message type in which the following exists:
 - The message contains a single part defined by an element.
 - There exists a catch activity with a matching `faultName` value that has a `faultVariable` whose associated `faultElement` QName matches the QName of the runtime element data of the single WSDL message part.

Then, the fault is sent to the identified catch activity with the `faultVariable` initialized to the value in the single part's element.

- Otherwise, if there is a catch activity with a matching `faultName` value that does not specify a `faultVariable` attribute, the fault is sent to the identified catch activity. In this case, the fault value is not available from within the fault handler, but is available to the rethrow activity.
- Otherwise, if there is a catch construct without a `faultName` attribute that has a `faultVariable` whose type matches the type of the runtime fault data, then the fault is sent to the identified catch activity.

- Otherwise, if the fault data is a WSDL message type in which the message contains a single part defined by an element and there exists a catch activity without a `faultName` attribute that has a `faultVariable` whose associated `faultElement`'s `QName` matches the `QName` of the runtime element data of the single WSDL message part, the fault is sent to the identified catch activity with the `faultVariable` initialized to the value in the single part's element.
- Otherwise, if there is a `catchAll` activity, the fault is sent to the `catchAll` fault handler.
- Otherwise, the fault is handled by the default fault handler.

11.3 Introduction to Categories of BPEL Faults

A BPEL fault has a fault name called a `Qname` (name qualified with a namespace) and a possible `messageType`. There are two categories of BPEL faults:

- Business faults
- Runtime faults

11.3.1 Business Faults

Business faults are application-specific faults that are generated when there is a problem with the information being processed (for example, when a social security number is not found in the database). A business fault occurs when an application executes a `throw` activity or when an `invoke` activity receives a fault as a response. The fault name of a business fault is specified by the BPEL process service component. The `messageType`, if applicable, is defined in the WSDL. A business fault can be caught with a `faultHandler` using the `faultName` and a `faultVariable`.

```
<catch faultName="ns1:faultName" faultVariable="varName">
```

11.3.2 Runtime Faults

Runtime faults are the result of problems within the running of the BPEL process service component or web service (for example, data cannot be copied properly because the variable name is incorrect). These faults are not user-defined, and are thrown by the system. They are generated if the process tries to use a value incorrectly, a logic error occurs (such as an endless loop), a Simple Object Access Protocol (SOAP) fault occurs in a SOAP call, an exception is thrown by the server, and so on.

Several runtime faults are automatically provided. These faults are included in the `http://schemas.oracle.com/bpel/extension` namespace. These faults are associated with the `messageType RuntimeFaultMessage`. The WSDL file shown in [Example 11-2](#) defines the `messageType`:

Example 11-2 *messageType Definition*

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="RuntimeFault"
  targetNamespace="http://schemas.oracle.com/bpel/extension"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="RuntimeFaultMessage">
    <part name="code" type="xsd:string" />
    <part name="summary" type="xsd:string" />
    <part name="detail" type="xsd:string" />
  </message>
</definitions>
```

```
</message>
</definitions>
```

If a `faultVariable` (of messageType `RuntimeFaultMessage`) is used when catching the fault, the fault code can be queried from the `faultVariable`, along with the fault summary and detail.

11.3.2.1 `bindingFault`

A `bindingFault` is thrown inside an activity if the preparation of the invocation fails. For example, the WSDL of the process fails to load. A `bindingFault` is not retryable. This type of fault usually must be fixed by human intervention.

11.3.2.2 `remoteFault`

A `remoteFault` is also thrown inside an activity. It is thrown because the invocation fails. For example, a SOAP fault is returned by the remote service.

11.3.2.3 `replayFault`

A `replayFault` replays the activity inside a scope. At any point inside a scope, this fault is migrated up to the scope. The server then re-executes the scope from the beginning.

11.4 Using the Fault Management Framework

Oracle SOA Suite provides a generic fault management framework for handling faults in BPEL processes. If a fault occurs during runtime in an invoke activity in a process, the framework catches the fault and performs a user-specified action defined in a fault policy file associated with the activity. If a fault results in a condition in which human intervention is the prescribed action, you perform recovery actions from Oracle Enterprise Manager Fusion Middleware Control. The fault management framework provides an alternative to designing a BPEL process with catch activities in scope activities.

This section provides an overview of the components that comprise the fault management framework.

- The fault management framework catches all faults (business and runtime) for an invoke activity.
- A fault policy file defines fault conditions and their corresponding fault recovery actions. Each fault condition specifies a particular fault or group of faults, which it attempts to handle, and the corresponding action for it. A set of actions is identified by an ID in the fault policy file.
- A set of conditions invokes an action (known as fault policy).
- A fault policy bindings file associates the policies defined in the fault policy file with the following:
 - SOA composite applications
 - BPEL process and Oracle Mediator service components
 - Reference binding components for BPEL process and Oracle Mediator service components

The framework looks for fault policy bindings in the same directory as the `composite.xml` file of the SOA composite application or in a remote location identified by two properties that you set.

Note: A fault policy configured with the fault management framework overrides any fault handling defined in catch activities of scope activities in the BPEL process. The fault management framework can be configured to rethrow the fault handling back to the catch activities.

- The fault policy file (`fault-policies.xml`) and fault policy bindings file (`fault-bindings.xml`) are placed in either of the following locations:
 - In the same directory as the `composite.xml` file of the SOA composite application.
 - In a different location that is specified with two properties that you add to the `composite.xml` file. This option is useful if a fault policy must be used by multiple SOA composite applications. This option overrides any fault policy files that are included in the same directory as the `composite.xml` file. [Example 11–3](#) provides details about these two properties. In this example, the fault policy files are placed into the SOA Metadata Service (MDS) shared area.

Example 11–3 Fault Policies used by Multiple SOA Composite Applications

```
<property
  name="oracle.composite.faultPolicyFile">orams:/apps/faultpolicyfiles/
  fault-policies.xml
</property>
<property
  name="oracle.composite.faultBindingFile">orams:/apps/faultpolicyfiles/
  fault-bindings.xml
</property>
```

See [Chapter 21, "Using Oracle Mediator Error Handling"](#) for details about Oracle Mediator fault handling capabilities.

11.4.1 How to Design a Fault Policy

This section describes how to design a fault policy.

Note: The Facades API enables you to programmatically perform the abort, retry (with a success action), continue, rethrow, and replay recovery options. For information, see *Oracle Fusion Middleware Infrastructure Management Java API Reference for Oracle SOA Suite*.

11.4.1.1 Understanding How Fault Policy Binding Resolution Works

A fault policy bindings file associates the policies defined in a fault policy file with the SOA composite application or the component (service component or reference binding component). The framework attempts to identify a fault policy binding in the following order:

- Reference binding component defined in the `composite.xml` file.
- BPEL process or Oracle Mediator service component defined in the `composite.xml` file.
- SOA composite application defined in the `composite.xml` file.

During the resolution process, if no action is found that matches the condition, the framework assumes that resolution failed and moves to the next resolution level.

For example, assume an invoke activity faults with `faultname="abc"`. There is a policy binding specified in the `fault-binding.xml` file:

- SOA composite application binds to `policy-id-1`
- BPEL process or Oracle Mediator service component or reference binding component binds to `policy-id-2`

In the `fault-bindings.xml` file, the following bindings are also specified:

- SOA composite application binds to `policy-id-3`
- Reference binding component or service component binds to `policy-id-4`

The fault management framework behaves as follows:

- First match the resolve binding (in this case, `policy-id-2`).
- If the fault resolution fails, go to the next possible match (`policy-id-4`).
- If the fault resolution fails, go to the next possible match (`policy-id-3`).
- If the fault resolution fails, go to the next possible match (in this case, `policy-id-1`).
- If the fault resolution still fails, the fault is sent to the BPEL fault catch activity.

11.4.1.2 Creating a Fault Policy File for Automated Fault Recovery

1. Create a fault policy file (for example, named `fault-policies.xml`). This file includes `condition` and `action` sections for performing specific tasks.
2. Place the file in the same directory as the `composite.xml` file or place it in a different location and define the `oracle.composite.faultPolicyFile` property. [Example 11-4](#) provides details.

Example 11-4 Defining Properties

```
<property
  name="oracle.composite.faultPolicyFile">orams:/apps/faultpolicyfiles/
  fault-policies.xml
</property>
<property
  name="oracle.composite.faultBindingFile">orams:/apps/faultpolicyfiles/
  fault-bindings.xml
</property>
```

3. Define the `condition` section of the fault policy file.
 - Note the following details about the `condition` section:
 - This section provides a condition based on `faultName`.
 - Multiple conditions may be configured for a `faultName`.
 - Each condition has one `test` section (an XPath expression) and one `action` section.
 - The `test` section (XPath expression) is evaluated for the fault variable available in the fault.
 - The `action` section has a reference to the action defined in the same file.
 - You can only query the fault variable available in the fault.

- The order of condition evaluation is determined by the sequential order in the document.

[Table 11–1](#) provides examples of condition section use in the fault policy file. All actions defined in the condition section must be associated with an action in the action section.

Table 11–1 Use of the condition Section in the Fault Policy File

Condition Example	Fault Policy File Syntax
This condition is checking a fault variable for code = "WSDLFailure"	<pre><condition> <test>\$fault.code="WSDLReading Error" </test></pre>
An action of ora-terminate is specified.	<pre><action ref="ora-terminate"/> </condition></pre>
No test condition is provided. This is a catchAll condition for a given faultName.	<pre><condition> <action ref="ora-rethrow"/> </condition></pre>
If the faultName name attribute is missing, this indicates a catchAll activity for faults that have any QName.	<pre><faultName > . . . </faultName></pre>

4. Define the action section of the fault policy file. Note that validation of fault policy files is done during deployment. If you change the fault policy, you must redeploy the SOA composite application that includes the fault policy.

[Table 11–2](#) provides several examples of action section use in the fault policy file. You can provide automated recovery actions for some faults. In all recovery actions except retry and human intervention, the framework performs the actions synchronously.

Table 11–2 Use of action Section in the Fault Policy File

Recovery Actions	Fault Policy File Syntax
Retry: Provides the following actions for retrying the activity. ■ Retry a specified number of times. ■ Provide a delay between retries (in seconds). ■ Increase the interval with an exponential back off. ■ Chain to a retry failure action if retry <i>N</i> times fails. ■ Chain to a retry success action if a retry is successful. Note: Exponential back off indicates the next retry attempt is scheduled at 2 x the <i>delay</i>, where <i>delay</i> is the current retry interval. For example, if the current retry interval is 2 seconds, the next retry attempt is scheduled at 4, the next at 8, and the next at 16 seconds until the <code>retryCount</code> value is reached.	<pre><Action id="ora-retry"> <Retry> <retryCount>3</retryCount> <retryInterval>2</retryInterval> <exponentialBackoff/> <retryFailureAction ref="ora-java"/> <retrySuccessAction ref="ora-java"/> </Retry> </Action></pre> Note the following details: ■ The framework chains to the retry success action if the retry attempt is successful. ■ If all retry attempts fail, the framework chains to the retry failure action.
Human Intervention: Causes the current activity to stop processing. You can now go to Oracle Enterprise Manager Fusion Middleware Control and perform manual recovery actions on this instance.	<pre><Action id="ora-human-intervention"> <humanIntervention/></Action></pre>
Terminate Process: Terminates the process	<pre><Action id="ora-terminate"><abort/></Action></pre>
Java Code: Enables you to execute an external Java class. <code>returnValue</code>: The implemented Java class must implement a method that returns a string. The policy can chain to a new action based on the returned string. For additional information, see Section 11.4.3, "How to Use a Java Action Fault Policy."	<pre><Action id="ora-java"> <!-- this is user provided custom java class--> <javaAction className="mypackage.myClass" defaultAction="ora-terminate"> <returnValue value="REPLAY" ref="ora-terminate"/> <returnValue value="RETHROW" ref="ora-rethrow-fault"/> <returnValue value="ABORT" ref="ora-terminate"/> <returnValue value="RETRY" ref="ora-retry"/> <returnValue value="MANUAL" ref="ora-human-intervention"/> </javaAction> </Action></pre>
Rethrow Fault: The framework sends the fault to the BPEL fault handlers (catch activities in scope activities). If none are available, the fault is sent up.	<pre><Action id="ora-rethrow-fault"><rethrowFault/></Action></pre>
Replay Scope: Raises a replay fault.	<pre><Action id="ora-replay-scope"><replayScope/></Action></pre>

Note: The preseeded recovery action tag names (ora-retry, ora-human-intervention, ora-terminate, and so on) are only samples. You can substitute these names with ones appropriate to your environment.

[Example 11-5](#) shows a fault policy file with fully-defined condition and action sections.

Notes:

- Fault policy file names are not restricted to one specific name. However, they must conform to the `fault-policy.xsd` schema file.
- [Example 11-5](#) provides an example of catching faults based on fault names. You can also catch faults based on message types, or on both:

```
<fault name="myfault" type="fault:faultType">
```

Example 11-5 Fault Policy File

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicies xmlns="http://schemas.oracle.com/bpel/faultpolicy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <faultPolicy version="0.0.1" id="FusionMidFaults"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.oracle.com/bpel/faultpolicy"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Conditions>
      <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
        <condition>
          <action ref="MediatorJavaAction"/>
        </condition>
      </faultName>
      <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:remoteFault">
        <condition>
          <action ref="BPELJavaAction"/>
        </condition>
      </faultName>
      <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:bindingFault">
        <condition>
          <action ref="BPELJavaAction"/>
        </condition>
      </faultName>
      <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:runtimeFault">
        <condition>
          <action ref="BPELJavaAction"/>
        </condition>
      </faultName>
    </Conditions>
    <Actions>
      <!-- Generics -->
```

```
<Action id="default-terminate">
  <abort/>
</Action>
<Action id="default-replay-scope">
  <replayScope/>
</Action>
<Action id="default-rethrow-fault">
  <rethrowFault/>
</Action>
<Action id="default-human-intervention">
  <humanIntervention/>
</Action>
<Action id="MediatorJavaAction">
  <!-- this is user provided class-->
  <javaAction className="MediatorJavaAction.myClass"
defaultAction="default-terminate">
    <returnValue value="MANUAL" ref="default-human-intervention"/>
  </javaAction>
</Action>
<Action id="BPELJavaAction">
  <!-- this is user provided class-->
  <javaAction className="BPELJavaAction.myAnotherClass"
defaultAction="default-terminate">
    <returnValue value="MANUAL" ref="default-human-intervention"/>
  </javaAction>
</Action>
</Actions>
</faultPolicy>
</faultPolicies>
```

11.4.1.3 Associating a Fault Policy with Fault Policy Binding

Note: The fault policy file binding file must be named `fault-bindings.xml`. This conforms to the `fault-bindings.xsd` schema file.

1. Create a fault policy binding file (`fault-bindings.xml`) that associates the policies defined in the fault policy file with the level of fault policy binding you are using (either a SOA composite application or a component (reference binding component or BPEL process or Oracle Mediator service component)).
2. Place the file in the same directory as the `composite.xml` file or place it in a remote location and define the `oracle.composite.faultBindingFile` property as shown in Step 2 of [Section 11.4.1.2, "Creating a Fault Policy File for Automated Fault Recovery."](#)

[Example 11–6](#) shows a fault policy bindings file that associates the fault policies defined in the `fault-policies.xml` file with the `FusionMidFaults` SOA composite application.

Example 11–6 *fault-buildings.xml* File

```
<?xml version="1.0" encoding="UTF-8" ?>
<faultPolicyBindings version="0.0.1"
  xmlns="http://schemas.oracle.com/bpel/faultpolicy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <composite faultPolicy="FusionMidFaults"/>
  <!--<composite faultPolicy="ServiceExceptionFaults"/>-->
```



```

    <!--<composite faultPolicy="GenericSystemFaults"/>-->
</faultPolicyBindings>

```

11.4.1.4 Additional Fault Policy and Fault Policy Binding File Samples

This section provides additional samples of fault policy and fault policy binding files. [Example 11-7](#) shows the `fault-policies.xml` file contents.

Example 11-7 *fault-policies.xml* File

```

<?xml version="1.0" encoding="UTF-8"?>
<faultPolicies xmlns="http://schemas.oracle.com/bpel/faultpolicy">
  <faultPolicy version="2.0.1"
    id="CRM_ServiceFaults"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.oracle.com/bpel/faultpolicy"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Conditions>
      <!-- Fault if wsdlRuntimeLocation is not reachable -->
      <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:remoteFault">
        <condition>
          <test>$fault.code="WSDLReadingError"</test>
          <action ref="ora-terminate"/>
        </condition>
        <condition>
          <action ref="ora-java"/>
        </condition>
      </faultName>
      <!-- Fault if location port is not reachable-->
      <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:bindingFault">
        <!--ORA-00001: unique constraint violated on insert-->
        <condition>
          <test>$fault.code="1"</test>
          <action ref="ora-java"/>
        </condition>
        <!--ORA-01400: cannot insert NULL -->
        <condition>
          <test xmlns:test="http://test">$fault.code="1400"</test>
          <action ref="ora-terminate"/>
        </condition>
        <!--ORA-03220: required parameter is NULL or missing -->
        <condition>
          <test>$fault.code="3220"</test>
          <action ref="ora-terminate"/>
        </condition>
        <condition>
          <action ref="ora-retry-crm-endpoint"/>
        </condition>
      </faultName>
      <!-- Business faults -->
      <!-- Fault comes with a payload of error, make sure the name space is
provided here or at root level -->
      <faultName xmlns:credit="http://services.otn.com"
name="credit:NegativeCredit">
        <!-- you get this fault when SSN starts with 0-->
        <condition>

```

```
        <test>$fault.payload="Bankruptcy Report"</test>
        <action ref="ora-human-intervention"/>
        <!--action ref="ora-retry"/-->
    </condition>
    <!-- you get this fault when SSN starts with 1-->
    <condition>
        <test>$fault.payload="Bankruptcy Report-abort"</test>
        <action ref="ora-terminate"/>
    </condition>
    <!-- you get this fault when SSN starts with 2-->
    <condition>
        <test>$fault.payload="Bankruptcy Report-rethrow"</test>
        <action ref="ora-rethrow-fault"/>
    </condition>
    <!-- you get this fault when SSN starts with 3-->
    <condition>
        <test>$fault.payload="Bankruptcy Report-replay"</test>
        <action ref="ora-replay-scope"/>
    </condition>
    <!-- you get this fault when SSN starts with 4-->
    <condition>
        <test
xmlns:myError="http://services.otn.com">$fault.payload="Bankruptcy
Report-human"</test>
        <action ref="ora-human-intervention"/>
    </condition>
    <!-- you get this fault when SSN starts with 5-->
    <condition>
        <test>$fault.payload="Bankruptcy Report-java"</test>
        <action ref="ora-java"/>
    </condition>
</faultName>

    </Conditions>
    <Actions>
        <Action id="ora-retry">
            <retry>
                <retryCount>3</retryCount>
                <retryInterval>2</retryInterval>
                <exponentialBackoff/>
                <retryFailureAction ref="ora-java"/>
                <retrySuccessAction ref="ora-java"/>
            </retry>
        </Action>
        <Action id="ora-retry-crm-endpoint">
            <retry>
                <retryCount>5</retryCount>
                <retryFailureAction ref="ora-java"/>
                <retryInterval>5</retryInterval>
                <retrySuccessAction ref="ora-java"/>
            </retry>
        </Action>
        <Action id="ora-replay-scope">
            <replayScope/>
        </Action>
        <Action id="ora-rethrow-fault">
            <rethrowFault/>
        </Action>
        <Action id="ora-human-intervention">
            <humanIntervention/>
        </Action>
    </Actions>
</Fault>
```

```

        </Action>
        <Action id="ora-terminate">
            <abort/>
        </Action>
        <Action id="ora-java">
            <!-- this is user provided class-->
            <javaAction
className="com.oracle.bpel.client.config.faultpolicy.TestJavaAction"
defaultAction="ora-terminate" propertySet="prop-for-billing">
                <returnValue value="REPLAY" ref="ora-terminate"/>
                <returnValue value="RETHROW" ref="ora-rethrow-fault"/>
                <returnValue value="ABORT" ref="ora-terminate"/>
                <returnValue value="RETRY" ref="ora-retry"/>
                <returnValue value="MANUAL" ref="ora-human-intervention"/>
            </javaAction>
        </Action>

        </Actions>
        <Properties>
            <propertySet name="prop-for-billing">
                <property name="user_email_recipient">bpeladmin</property>
                <property name="email_recipient">joe@abc.com</property>
                <property name="email_recipient">mike@xyz.com</property>
                <property name="email_threshold">10</property>
                <property name="sms_recipient">+429876547</property>
                <property name="sms_recipient">+4212345</property>
                <property name="sms_threshold">20</property>
                <property name="user_email_recipient">john</property>
            </propertySet>
            <propertySet name="prop-for-order">
                <property name="email_recipient">john@abc.com</property>
                <property name="email_recipient">jill@xyz.com</property>
                <property name="email_threshold">10</property>
                <property name="sms_recipient">+42222</property>
                <property name="sms_recipient">+423335</property>
                <property name="sms_threshold">20</property>
            </propertySet>
        </Properties>
    </faultPolicy>
    <faultPolicy version="2.0.1"
        id="Billing_ServiceFaults"
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"

        xmlns="http://schemas.oracle.com/bpel/faultpolicy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

        <Conditions>
            <faultName>
                <condition>
                    <action ref="ora-manual"/>
                </condition>
            </faultName>
        </Conditions>
        <Actions>
            <Action id="ora-manual">
                <humanIntervention/>
            </Action>
        </Actions>
    </faultPolicy>

```

```
</faultPolicies>
```

[Example 11-8](#) shows the `fault-buildings.xml` file that associates the fault policies defined in `fault-policies.xml`.

Example 11-8 Fault Policy Bindings File

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicyBindings version="2.0.1"
  xmlns="http://schemas.oracle.com/bpel/faultpolicy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <composite faultPolicy="ConnectionFaults"/>
  <component faultPolicy="ServiceFaults">
    <name>Component1</name>
    <name>Component2</name>
  </component>
  <!-- Below listed component names use polic CRM_ServeFaults -->
  <component faultPolicy="CRM_ServiceFaults">
    <name>HelloWorld</name>
    <name>ShippingComponent</name>
    <name>AnotherComponent"</name>
  </component>
  <!-- Below listed reference names and port types use polic CRM_ServiceFaults
  -->
  <reference faultPolicy="CRM_ServiceFaults">
    <name>creditRatingService</name>
    <name>anotherReference</name>
    <portType
  xmlns:credit="http://services.otn.com">credit:CreditRatingService</portType>
    <portType
  xmlns:db="http://xmlns.oracle.com/pcbpel/adapter/db/insert/">db:insert_
  plt</portType>
    </reference>
    <reference faultPolicy="test1">
      <name>CreditRating3</name>
    </reference>
  </faultPolicyBindings>
```

11.4.1.5 Designing a Fault Policy with Multiple Rejection Handlers

If you design a fault policy that uses the action handler for rejected messages, note that only one write action can be performed. Multiple write actions cannot be performed, even if you define multiple rejection handlers, as shown in [Example 11-9](#). In this case, only the first rejection handler defined (for this example, `ora-queue`) is executed.

Example 11-9 Fault Policy with Multiple Rejection Handlers

```
<faultName xmlns:rjm="http://schemas.oracle.com/sca/rejectedmessages"
  name="rjm:FileIn">
  <condition>
    <action ref="ora-queue"/>
  </condition>
</faultName>
<faultName xmlns:rjm="http://schemas.oracle.com/sca/rejectedmessages"
  name="rjm:FileIn">
  <condition>
    <action ref="ora-file"/>
  </condition>
```

```
</faultName>
```

11.4.2 How to Execute a Fault Policy

You deploy a fault policy as part of a SOA composite application. After deployment, you can perform the following fault recovery actions from Oracle Enterprise Manager Fusion Middleware Control:

- Retry the activity
- Modify a variable (available to the faulted activity)
- Continue the instance (mark the activity as a success)
- Rethrow the exception
- Abort the instance
- Throw a replay scope exception

For additional information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for the following:

- Instructions on executing a fault policy in Oracle Enterprise Manager Fusion Middleware Control
- Use cases in which you define a fault policy that uses human intervention

11.4.3 How to Use a Java Action Fault Policy

Note the following details when using the Java action fault policy:

- The Java class provided follows a specific interface. This interface returns a string. Multiple values can be provided for output and fault policy to take after execution.
- Additional fault policy can be executed by providing a mapping from the output value (return value) of implemented methods to a fault policy.
- If no ReturnValue is specified, the default fault policy is executed, as shown in [Example 11-10](#).

Example 11-10 Java Action Fault Policy

```
<Action id="ora-java">
  <javaAction className="mypackage.myclass"
    defaultAction="ora-human-intervention" propertySet="prop-for-billing">
    <!--defaultAction is a required attribute, but propertySet is optional-->
    <!-- attribute-->
      <ReturnValue value="RETRY" ref="ora-retry"/>
      <!--value is not nilable attribute & cannot be empty-->
      <ReturnValue value="RETHROW" ref="ora-rethrow-fault"/>
    </javaAction>
  </Action>
```

[Table 11-3](#) provides an example of ReturnValue use.

Table 11-3 System Interpretation of Java Action Fault Policy

Code	Description
<ReturnValue value="RETRY" ref="ora-retry" />	Execute the ora-retry action if the method returns a string of RETRY.

Table 11–3 (Cont.) System Interpretation of Java Action Fault Policy

Code	Description
<code><ReturnValue value="" ref="ora-rethrow"/></code>	Fails in validation.
<code><javaAction className="mypackage.myclass" defaultAction="ora-human-intervention" ></code>	Execute ora-human-intervention after Java code execution. This attribute is used if the return from the method does not match any provided ReturnValue.
<code><ReturnValue value="RETRY" ref="ora-retry"/> <ReturnValue value="" ref="" /></code>	Fails in validation.
<code><javaAction className="mypackage.myclass" defaultAction=" ora-human-intervention"> <ReturnValue></ReturnValue></code>	Fails in validation.

To invoke a Java class, you can provide a class that implements the `IFaultRecoveryJavaClass` interface. `IFaultRecoveryJavaClass` is included in the `fabric-runtime.jar` file. The package name is `oracle.integration.platform.faultpolicy`.

The `IFaultRecoveryJavaClass` interface has two methods, as shown in [Example 11–11](#).

Example 11–11 implementation of `IFaultRecoveryJavaClass`

```
public interface IFaultRecoveryJavaClass
{
    public void handleRetrySuccess( IFaultRecoveryContext ctx );
    public String handleFault( IFaultRecoveryContext ctx );
}
```

Note the following details:

- `handleRetrySuccess` is invoked upon a successful retry attempt. The retry policy chains to a Java action on `retrySuccessAction`.
- `handleFault` is invoked to execute a policy of type `javaAction`.

[Example 11–12](#) shows the data available with `IFaultRecoveryContext`:

Example 11–12 Data Available with `IFaultRecoveryContext`

```
public interface IFaultRecoveryContext {

    /**
     * Gets implementation type of the fault.
     * @return
     */
    public String getType();

    /**
     * @return Get property set of the fault policy action being executed.
     */
    public Map getProperties();
}
```

```

/**
 * @return Get fault policy id of the fault policy being executed.
 */
public String getPolicyId();

/**
 * @return Name of the faulted partner link.
 */
public String getReferenceName();

/**
 * @return Port type of the faulted reference .
 */
public QName getPortType();
}

```

The service engine implementation of this interface provides more information (for example, Oracle BPEL Process Manager). [Example 11-13](#) provides details.

Example 11-13 Service Engine Implementation of IFaultRecoveryContext

```

public class BPELFaultRecoveryContextImpl extends BPELXExecLetUtil implements
IBPELFaultRecoveryContext, IFaultRecoveryContext{
...
}

```

Oracle BPEL Process Manager-specific data is available with IBPELFaultRecoveryContext, as shown in [Example 11-14](#).

Example 11-14 Oracle BPEL Process Manager-Specific Data

```

public interface IBPELFaultRecoveryContext {
public void addAuditTrailEntry(String message);

public void addAuditTrailEntry(String message, Object detail);

public void addAuditTrailEntry(Throwable t);
/**
 * @return Get action id of the fault policy action being executed.
 */
public String getActionId();

/**
 * @return Type of the faulted activity.
 */
public String getActivityId();

/**
 * @return Name of the faulted activity.
 */
public String getActivityName();

/**
 * @return Type of the faulted activity.
 */
public String getActivityType();

/**
 * @return Correlation id of the faulted activity.
 */
}

```

```
public String getCorrelationId();

/**
 * @return BPEL fault that caused the invoke to fault.
 */
public BPELFault getFault();

/**
 * @return Get index value of the instance
 */
public String getIndex(int i);

/**
 * @return get Instance Id of the current process instance of the faulted
 *         activity.
 */
public long getInstanceId();

/**
 * @return Get priority of the current process instance of the faulted
 *         activity.
 */
public int getPriority();

/**
 * @return Process DN.
 */
public ComponentDN getProcessDN();

/**
 * @return Get status of the current process instance of the faulted
 *         activity.
 */
public String getStatus();

/**
 * @return Get title of the current process instance of the faulted
 *         activity.
 */
public String getTitle();

public Object getVariableData(String name) throws BPELFault;

public Object getVariableData(String name, String partOrQuery)
throws BPELFault;

public Object getVariableData(String name, String part, String query)
throws BPELFault;

/**
 * @param priority
 *         Set priority of the current process instance of the faulted
 *         activity.
 * @return
 */
public void setPriority(int priority);

/**
 * @param status
 *         Set status of the current process instance of the faulted
```



```

        *          activity.
    */
    public void setStatus(String status);

    /**
     * @param title
     *          Set title of the current process instance of the faulted
     *          activity.
     * @return
     */
    public String setTitle(String title);

    public void setVariableData(String name, Object value) throws BPELFault;

    public void setVariableData(String name, String partOrQuery, Object value)
        throws BPELFault;

    public void setVariableData(String name, String part, String query,
        Object value) throws BPELFault;
}

```

[Example 11–15](#) provides an example of `javaAction` implementation.

Example 11–15 Implementation of a `javaAction`

```

public class TestJavaAction implements IFaultRecoveryJavaClass {
    public void handleRetrySuccess(IFaultRecoveryContext ctx) {
        System.out.println("This is for retry success");
        handleFault(ctx);
    }
    public String handleFault(IFaultRecoveryContext ctx) {
        System.out.println("-----Inside handleFault-----\n" + ctx.toString());

        dumpProperties(ctx.getProperties());
        /* Get BPEL specific context here */
        BPELFaultRecoveryContextImpl bpelCtx = (BPELFaultRecoveryContextImpl) ctx;
        bpelCtx.addAuditTrailEntry("hi there");
        System.out.println("Policy Id" + ctx.getPolicyId());
        ...
    }
}

```

11.4.4 What You May Need to Know About Fault Management Behavior When the Number of Instance Retries is Exceeded

When you configure a fault policy to recover instances with the `ora-retry` action and the number of specified instance retries is exceeded, the instance is marked as `open.faulted` (in-flight state). The instance remains active.

Marking instances as `open.faulted` ensures that no instances are lost. You can then configure another fault handling action following the `ora-retry` action in the fault policy file, such as the following:

- Configure an `ora-human-intervention` action to manually perform instance recovery from Oracle Enterprise Manager Fusion Middleware Control.
- Configure an `ora-terminate` action to close the instance (mark it as `closed.faulted`) and never retry again.

However, if you do *not* set an action to be performed after an `ora-retry` action in the fault policy file and the number of instance retries is exceeded, the instance *remains* marked as `open.faulted`, and recovery attempts to handle the instance.

For example, if no action is defined in the fault policy file shown in [Example 11–16](#) after `ora-retry`:

Example 11–16 No Action Defined

```
<Action id="ora-retry">
  <retry>
    <retryCount>2</retryCount>
    <retryInterval>2</retryInterval>
    <exponentialBackoff/>
  </retry>
</Action>
```

The following actions are performed:

- The invoke activity is attempted (using the above-mentioned fault policy code to handle the fault).
- Two retries are attempted at increasing intervals (after two seconds, then after four seconds).
- If all retry attempts fail, the following actions are performed:
 - A detailed fault error message is logged in the audit trail.
 - The instance is marked as `open.faulted` (in-flight state).
 - The instance is picked up and the invoke activity is re-attempted.
- Recovery may also fail. In that case, the invoke activity is re-executed. Additional audit messages are logged.

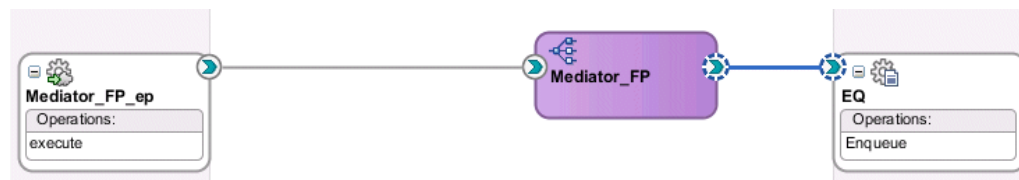
11.4.5 What You May Need to Know Executing the Retry Action with Multiple Faults in the Same Flow

The fault policy retry action may not execute with multiple faults in the same flow. This may be because the retry count has already been reached for any of the previous faults.

For example, assume you define a fault policy with two fault conditions: `fault1` and `fault2`. For both fault conditions, the retry action is specified with a retry count of three. Assume `fault1` occurs and the retry action executes three times. You correct the problem for `fault1` by modifying the payload, but ensure that `fault2` is to be raised when the instance is resubmitted. You then resubmit the faulted instance using Oracle Enterprise Manager Fusion Middleware Control. You expect the second fault condition, `fault2`, to retry three times according to the fault policy specification. However, this does not occur because the maximum number of retries was already executed for the previous `fault1` fault condition.

11.4.6 What You May Need to Know About Binding Level Retry Execution Within Fault Policy Retries

If you are testing retry actions on adapters with both JCA-level retries for the outbound direction and a retry action in the fault policy file for outbound failures, the JCA-level (or binding level) retries are executed within the fault policy retries. For example, assume you have designed the application shown in [Figure 11–2](#):

Figure 11–2 SOA Composite Application

You specify the retry parameters shown in [Example 11–17](#) in the `composite.xml` file:

Example 11–17 Retry Parameters

```

<property name="jca.retry.count" type="xs:int" many="false"
  override="may">2</property>
<property name="jca.retry.interval" type="xs:int" many="false"
  override="may">2</property>
<property name="jca.retry.backoff" type="xs:int" many="false"
  override="may">2</property>

```

In the fault policy file for the EQ reference binding component for the outbound direction, you specify the actions shown in [Example 11–18](#).

Example 11–18 Retry Actions

```

<retryCount>3</retryCount>
<retryInterval>3</retryInterval>

```

If an outbound failure occurs, the expected behavior is for the JCA retries to occur within the fault policy retries. When the first retry of the fault policy is executed, the JCA retry is called. In this example, a JCA retry of 2 with an interval of 2 seconds and exponential back off of 2 is executed for every retry of the fault policy:

- Fault policy retry 1:
 - JCA retry 1 (with 2 seconds interval)
 - JCA retry 2 (with 4 seconds interval)
- Fault policy retry 2:
 - JCA retry 1 (with 2 seconds interval)
 - JCA retry 2 (with 4 seconds interval)
- Fault policy retry 3:
 - JCA retry 1 (with 2 seconds interval)
 - JCA retry 2 (with 4 seconds interval)

11.4.7 What You May Need to Know About Defining the `ora-java` Option

Assume you invoke a SOA composite application with a fault policy/binding defined and see a recoverable fault in Oracle Enterprise Manager Fusion Middleware Control. After you perform a successful fault recovery retry, note that there is no **ora-java** option available for selection by default in the **After Successful Retry** list of the **Faults** tab of the Instance of `process_name` page.

This is the expected behavior. For the **ora-java** option to display, you must explicitly define it in the `fault-policies.xml` file during design-time. For example, perform the following steps.

1. Create a `fault-policies.xml` file in which you explicitly add `retrySuccessAction ref="ora-java"/>` to the `fault-policies.xml` file.

```
<Action id="ora-retry">
  <Retry>
    <retryCount>3</retryCount>
    <retryInterval>2</retryInterval>
    <exponentialBackoff/>
    <retryFailureAction ref="ora-java"/>
    <retrySuccessAction ref="ora-java"/>
  </Retry>
</Action>
```

2. Deploy the composite and create an instance.
3. Click the composite instance to invoke the instance trace of the composite.
4. Click the component in which there is a recoverable fault (for example, Oracle BPEL Process Manager, Oracle Mediator, or Oracle BPM).
5. Go to the **Faults** tab.
6. Select the **Retry** option to successfully retry the fault.

If fault recovery is successful, the **After Successful Retry** list is displayed.

7. Select the list and note that the **ora-java** option is now listed.

For more information about recovering from faults in Oracle Enterprise Manager Fusion Middleware Control, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

11.5 Catching BPEL Runtime Faults

BPEL runtime faults can be caught as a named BPEL fault. The `bindingFault` and `remoteFault` can be associated with a message. This action enables the `faultHandler` to get details about the faults.

11.5.1 How to Catch BPEL Runtime Faults

The following procedure shows how to use the provided examples to generate a fault and define a fault handler to catch it. In this case, you modify a WSDL file to generate a fault, and create a catch attribute to catch it.

To catch BPEL runtime faults:

1. Import `RuntimeFault.wsdl` into your process WSDL. `RuntimeFault.wsdl` is seeded into the MDS from `soa.mar` inside `soa-infra-wls.ear` during its deployment.

You may see a copy of `soa.mar` in the deployed SOA Infrastructure in the Oracle WebLogic Server domain, which is a JAR/ZIP file containing `RuntimeFault.wsdl`.

2. Declare a variable with `messageType bpelx:RuntimeFaultMessage`.
3. Catch it using the following syntax:

```
<catch faultName="bpelx:remoteFault" | "bpelx:bindingFault"
  faultName="varName">
```

11.6 Getting Fault Details with the getFaultAsString XPath Extension Function

The catchAll activity is provided to catch possible faults. However, BPEL does not provide a method for obtaining additional information about the captured fault. Use the `getFaultAsString()` XPath extension function to obtain additional information.

11.6.1 How to Get Fault Details with the getFaultAsString XPath Extension Function

[Example 11–19](#) shows how to use this function.

Example 11–19 `getFaultAsString()` XPath Extension Function

```
<catchAll>
  <sequence>
    <assign>
      <from expression="bpelx:getFaultAsString()" />
      <to variable="faultVar" part="message" />
    </assign>
    <reply faultName="ns1:myFault" variable="faultVar" .../>
  </sequence>
</catchAll>
```

11.7 Throwing Internal Faults

A BPEL application can generate and receive fault messages. The throw activity has three elements: its name, the name of the fault, and the fault variable. 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. Throw activity syntax includes the throw name, fault name, and fault variable:

```
<throw name="delay" faultName="nsPrefix:fault-1" faultVariable="fVar" />
```

11.7.1 How to Create a Throw Activity

To create a throw activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag a **Throw** activity into the designer.
3. Double-click and define the **Throw** activity.
4. Optionally enter a name or accept the default value.
5. To the right of the **Namespace URI** field, click the **Search** icon to select the fault to monitor.
6. Select the fault in the Fault Chooser dialog, and click **OK**.

The namespace URI for the selected fault displays in the **Namespace URI** field. Your fault selection also automatically displays in the **Local Part** field.

[Figure 11–3](#) provides an example of a completed Throw dialog. This example shows the **Throw_Fault_CC_Denied** throw activity of the **Scope_AuthorizeCreditCard** scope activity in the Fusion Order Demo application. This activity throws a fault for orders that are not approved.

Figure 11–3 Throw Dialog

The 'Throw' dialog box is shown with the following details:

- General Tab:**
 - Name:** Throw_Fault_CC_Denied
 - Fault QName:**
 - Namespace URI: v.globalcompany.example.com/ns/OrderBookingService
 - Local Part: OrderProcessorFault
 - Fault Variable:** gOrderProcessorFaultVariable
- Buttons:** Help, Apply, OK, Cancel

7. Click **Apply**, then **OK**.

11.7.2 What Happens When You Create a Throw Activity

Example 11–20 shows the throw activity in the .bpel file after design completion. The OrderProcessor process terminates after executing this throw activity.

Example 11–20 Throw Activity

```
<throw name="Throw_Fault_CC_Denied"
      faultName="client:OrderProcessorFault"/>
```

11.8 Rethrowing Faults with the Rethrow Activity

The rethrow activity rethrows faults originally captured by the immediately enclosing fault handler. Only use the rethrow activity within a fault handler (for example, within catch and catchAll activities). The rethrow activity is used in fault handlers to rethrow the captured fault (that is, the fault name and the fault data (if present) of the original fault). The rethrow activity must ignore modifications to fault data. For example:

- If the fault handler modifies fault data and then calls a rethrow activity, the original fault data is rethrown, and not the modified fault data.
- If a fault is captured using the functionality that enables message type faults with one part defined using an element to be caught by fault handlers looking for the same element type, then the rethrow activity rethrows the original message type data.

Note: This activity is supported in BPEL version 2.0 projects.

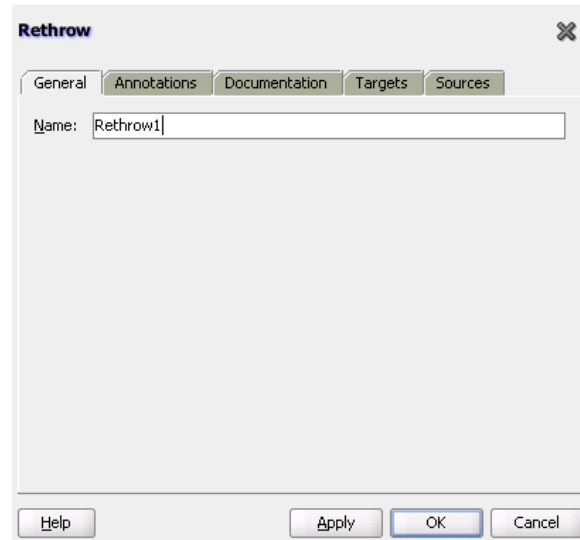
11.8.1 How to Create a Rethrow Activity

To create a rethrow activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag a **Rethrow** activity into the designer.

3. Double-click and define the **Rethrow** activity.
4. Optionally enter a name or accept the default value, as shown in [Figure 11-4](#).

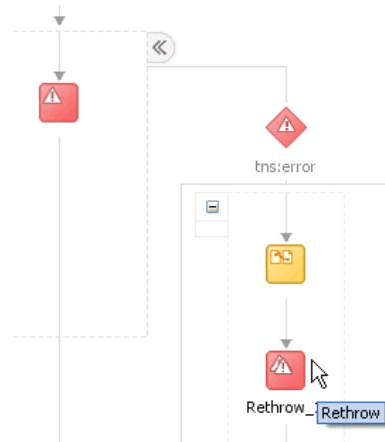
Figure 11-4 Rethrow Dialog



5. Click **Apply**, then **OK**.

When complete, design can look similar to that shown in [Figure 11-5](#).

Figure 11-5 Throw Activity in BPEL Process



11.8.2 What Happens When You Rethrow Faults

[Example 11-21](#) shows the .bpe1 file after design is complete for a rethrow activity. The rethrow activity is inside a fault handler (catch activity).

Example 11-21 Rethrow Activity

```
<scope name="scope1">
  <faultHandlers>
    <catch faultName="tns:error" faultVariable="tmpVar"
      faultElement="tns:fault">
      <sequence>
```

```
<assign>
  <copy>
    <from>concat('caught fault: ', $tmpVar)</from>
    <to>$output.payload</to>
  </copy>
</assign>
<rethrow name="Rethrow_1"/>
</sequence>
</catch>
</faultHandlers>
<throw faultName="tns:error" faultVariable="fault"/>
</scope>
```

11.9 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.

11.9.1 How to Return a Fault in a Synchronous Interaction

The syntax of a reply activity that returns a fault in a synchronous interaction is shown in [Example 11-22](#):

Example 11-22 Reply Activity

```
<reply partnerlink="partner-link-name"
  portType="port-type-name"
  operation="operation-name"
  variable="variable-name" (optional)
  faultName="fault-name">
</reply>
```

Always returning a fault in response to a synchronous request is not very useful. It is better to make the activity part of a conditional branch, in which the first branch is executed if the data requested is available. If the requested data is not available, then the BPEL process service component returns a fault with this information.

For more information, see the following chapters:

- [Chapter 7, "Invoking a Synchronous Web Service from a BPEL Process"](#) for synchronous interactions
- [Chapter 10, "Using Conditional Branching in a BPEL Process"](#) for setting up the conditional structure

11.9.2 How to Return a Fault in an Asynchronous Interaction

In an asynchronous interaction, the client does not wait for a reply. The reply activity is not used to return a fault. Instead, the BPEL process service component returns a fault using a callback operation on the same port type that normally receives the requested information, with an invoke activity.

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

11.10 Using a Scope Activity to Manage a Group of Activities

A scope activity provides a container and a context for other activities. A scope provides handlers for faults, events, compensation, data variables, and correlation sets. Using a scope activity simplifies a BPEL flow by grouping functional structures. This grouping enables you to collapse them into what appears to be a single element in Oracle BPEL Designer.

[Example 11–23](#) shows a scope named `Scope_FulfillOrder` from the WebLogic Fusion Order Demo application. This scope invokes the `FulfillOrder` Oracle Mediator component, which determines the shipping method for the order.

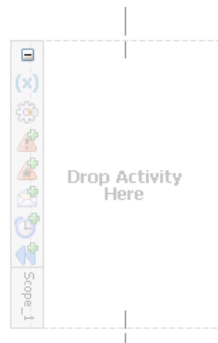
Example 11–23 Scope Activity

```
<scope name="Scope_FulfillOrder">
  <variables>
    <variable name="lFulfillOrder_InputVariable"
      messageType="ns17:requestMessage" />
  </variables>
  <sequence>
    <assign name="Assign_OrderData">
      <copy>
        <from variable="gOrderInfoVariable"
          query="/ns4:orderInfoVOSDO" />
        <to variable="lFulfillOrder_InputVariable"
          part="request" query="/ns4:orderInfoVOSDO" />
      </copy>
    </assign>
    <invoke name="Invoke_FulfillOrder"
      inputVariable="lFulfillOrder_InputVariable"
      partnerLink="FulfillOrder.FulfillOrder"
      portType="ns17:execute_ptt" operation="execute" />
  </sequence>
</scope>
```

11.10.1 How to Create a Scope Activity

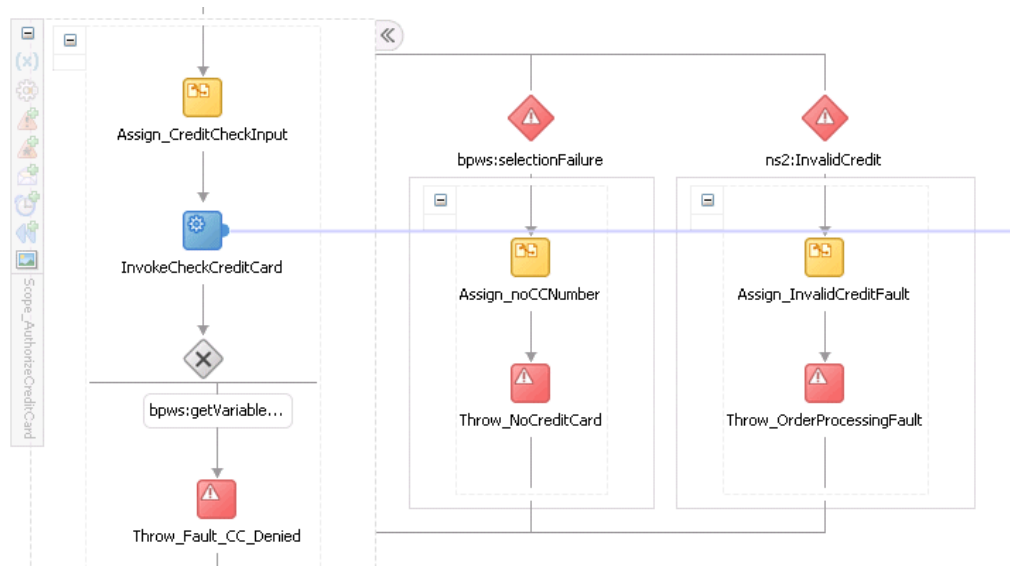
To create a scope activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag a **Scope** activity into the designer.
3. Open the **scope** activity by double-clicking it or by single-clicking the **Expand** icon.
4. From the Component Palette, drag and define activities to build the functionality within the scope. [Figure 11–6](#) provides details.

Figure 11–6 Expanded Scope Activity

5. Click **OK**.

When complete, scope activity design can look as shown in [Figure 11–7](#). This example shows the **Scope_AuthorizeCreditCard** scope activity of the Fusion Order Demo application.

Figure 11–7 Scope Activity After Design Completion

11.10.2 How to Add Descriptive Notes and Images to a Scope Activity

You can add descriptive notes to scope activities that provide simple descriptions of the functionality of the scope. You can also change the graphical image of scopes. The notes and images display in Oracle BPEL Designer. This helps to make a scope easier to understand.

To add descriptive notes and images to a scope activity:

1. Perform one of the following steps:
 - Right-click the scope and select **User Documentation**.
 - Double-click the scope and select the **User Documentation** tab.

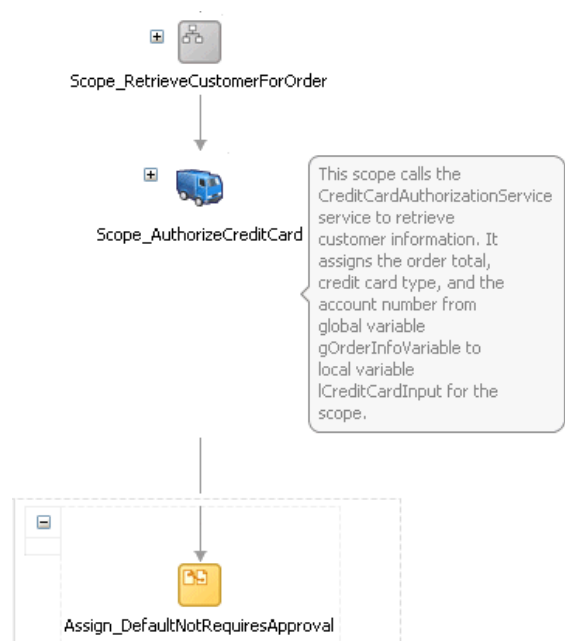
The Documentation dialog appears.

2. In the **Comment** field, enter a brief description of the functionality of the scope.

3. In the **Image** field, click the **Search** icon to optionally change the graphical image for the scope.
4. Click **OK**.

Your changes display in Oracle BPEL Designer, as shown in [Figure 11–8](#).

Figure 11–8 Scope with Descriptive Note and Modified Image



5. To edit the note, double-click it.

11.10.3 What Happens After You Create a Scope Activity

[Example 11–24](#) shows the scope activity in the `.bpel` file after design completion. The `Scope_AuthorizeCreditCard` scope activity consists of activities that perform the following actions:

- A catch activity for catching faulted orders in which the credit card number is not provided or the credit type is not valid.
- A throw activity that throws a fault for orders that are not approved.
- An assign activity that takes the credit card type, credit card number, and purchase amount, and assigns this information to the input variable for the `CreditCardAuthorizationService` service.
- An invoke activity that calls a `CreditCardAuthorizationService` service to retrieve customer information.
- A switch activity that checks the results of the credit card validation.

Example 11–24 Scope Activity

```
<scope name="Scope_AuthorizeCreditCard">
  <variables>
    <variable name="lCreditCardInput"
      messageType="ns2:CreditAuthorizationRequestMessage" />
    <variable name="lCreditCardOutput"
      messageType="ns2:CreditAuthorizationResponseMessage" />
  </variables>
  <body>
    <!-- Body content -->
  </body>
</scope>
```

```
</variables>
<faultHandlers>
  <catch faultName="bpws:selectionFailure">
    <sequence>
      <assign name="Assign_noCCNumber">
        <copy>
          <from expression="string('CreditCardCheck - NO
            CreditCard')"/>
          <to variable="gOrderProcessorFaultVariable"
            part="code"/>
        </copy>
      </assign>
      <throw name="Throw_NoCreditCard"
        faultVariable="gOrderProcessorFaultVariable"
        faultName="ns9:OrderProcessingFault"/>
    </sequence>
  </catch>
  <catch faultName="ns2:InvalidCredit">
    <sequence>
      <assign name="Assign_InvalidCreditFault">
        <copy>
          <from expression="concat(bpws:getVariableData
            ('gOrderInfoVariable','/ns4:orderInfoVOSDO/
            ns4:CardTypeCode'),' is not a valid
            creditcard type')"/>
          <to variable="gOrderProcessorFaultVariable"
            part="summary"/>
        </copy>
        <copy>
          <from expression="string('CreditCardCheck - NOT VALID')"/>
          <to variable="gOrderProcessorFaultVariable"
            part="code"/>
        </copy>
      </assign>
      <throw name="Throw_OrderProcessingFault"
        faultName="ns9:OrderProcessingFault"
        faultVariable="gOrderProcessorFaultVariable"/>
    </sequence>
  </catch>
</faultHandlers>
<sequence>
  <assign name="Assign_CreditCheckInput">
    <copy>
      <from variable="gOrderInfoVariable"
        query="/ns4:orderInfoVOSDO/ns4:OrderTotal"/>
      <to variable="lCreditCardInput" part="Authorization"
        query="/ns8:AuthInformation/ns8:PurchaseAmount"/>
    </copy>
    <copy>
      <from variable="gOrderInfoVariable"
        query="/ns4:orderInfoVOSDO/ns4:CardTypeCode"/>
      <to variable="lCreditCardInput" part="Authorization"
        query="/ns8:AuthInformation/ns8:CCType"/>
    </copy>
    <copy>
      <from variable="gOrderInfoVariable"
        query="/ns4:orderInfoVOSDO/ns4:AccountNumber"/>
      <to variable="lCreditCardInput" part="Authorization"
        query="/ns8:AuthInformation/ns8:CCNumber"/>
    </copy>
  </assign>
</sequence>
```

```

</assign>
<invoke name="InvokeCheckCreditCard"
  inputVariable="lCreditCardInput"
  outputVariable="lCreditCardOutput"
  partnerLink="CreditCardAuthorizationService"
  portType="ns2:CreditAuthorizationPort"
  operation="AuthorizeCredit"/>
<switch name="Switch_EvaluateCCResult">
  <case condition="bpws:getVariableData('lCreditCardOutput','status','
    /ns8:status') != 'APPROVED'">
    <bpelx:annotation>
      <bpelx:pattern>status &lt;&gt; approved</bpelx:pattern>
    </bpelx:annotation>
    <throw name="Throw_Fault_CC_Denied"
      faultName="client:OrderProcessorFault"/>
  </case>
</switch>
</sequence>
</scope>

```

11.10.4 What You May Need to Know About Scopes

Scopes can use a significant amount of CPU and memory and should not be overused. Sequence activities use less CPU and memory and can make large BPEL flows more readable.

11.10.5 How to Use a Fault Handler Within a Scope

If a fault is not handled, it creates a faulted state that migrates up through the application and can throw the entire process into a faulted state. To prevent this from occurring, place the parts of the process that have the potential to receive faults within a scope. The scope activity includes the following fault handling capabilities:

- The catch activity works within a scope to catch faults and exceptions before they can throw the entire process into a faulted state. You can use specific fault names in the catch activity to respond in a specific way to an individual fault.
- The catchAll activity catches any faults that are not handled by name-specific catch activities.

Example 11–25 shows the syntax for catch and catchAll activities. Assume that a fault named `x:foo` is thrown. The first catch is selected if the fault carries no fault data. If there is fault data associated with the fault, the third catch is selected if the type of the fault's data matches the type of variable `bar`. Otherwise, the default catchAll handler is selected. Finally, a fault with a fault variable whose type matches the type of `bar` and whose name is not `x:foo` is processed by the second catch. All other faults are processed by the default catchAll handler.

Example 11–25 Catch and CatchAll Activities

```

<faulthandlers>
  <catch faultName="x:foo">
    <empty/>
  </catch>
  <catch faultVariable="bar">
    <empty/>
  </catch>
  <catch faultName="x:foo" faultVariable="bar">
    <empty/>
  </catch>

```

```

<catchAll>
  <empty/>
</catchAll>
</faulthandlers>

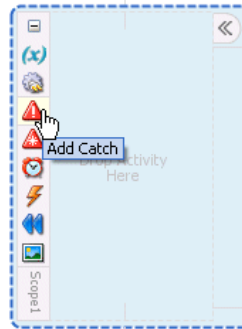
```

11.10.6 How to Create a Catch Activity in a Scope

To create a catch activity in a scope:

1. In the expanded **Scope** activity, click **Add Catch**. [Figure 11–9](#) provides details.

Figure 11–9 Add Catch

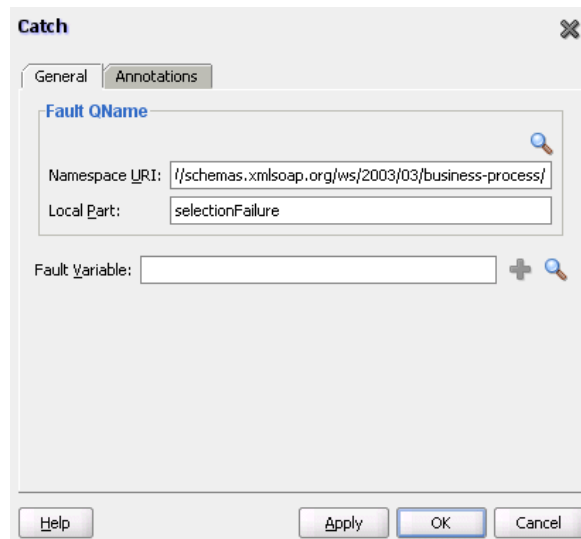


This creates a catch activity in the right side of the scope activity.

2. Double-click the **Catch** activity.
3. Optionally enter a name.
4. To the right of the **Namespace URI** field, click the **Search** icon to select the fault.
5. Select the fault in the Fault Chooser dialog, and click **OK**.

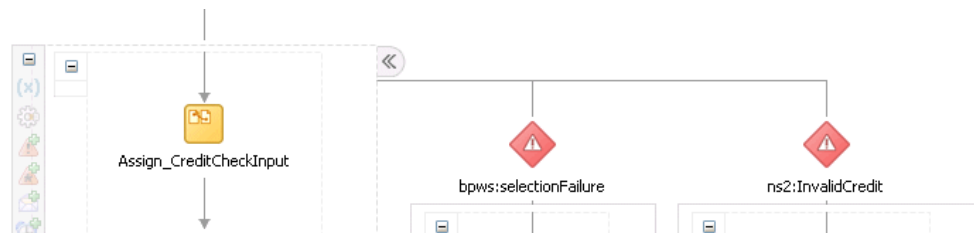
The namespace URI for the selected fault displays in the **Namespace URI** field. Your fault selection also automatically displays in the **Local Part** field.

[Figure 11–10](#) provides an example of a Catch dialog. This example shows the **selectionFailure** catch activity of the **Scope_AuthorizeCreditCard** scope activity in the Fusion Order Demo application. This catch activity catches orders in which the credit card number is not provided.

Figure 11–10 *Catch Dialog*

6. Design additional fault handling functionality.
7. Click OK.

Figure 11–11 provides an example of two catch activities for the **Scope_AuthorizeCreditCard** scope activity. The second catch activity catches credit types that are not valid.

Figure 11–11 *Catch Activities in the Designer*

11.10.7 What Happens When You Create a Catch Activity in a Scope

Example 11–26 shows the catch activity in the .bpel file after design completion. The `selectionFailure` catch activity catches orders in which the credit card number is not provided and the `InvalidCredit` catch activity catches credit types that are not valid.

Example 11–26 *Catch Branch*

```
<faultHandlers>
  <catch faultName="bpws:selectionFailure">
    <sequence>
      <assign name="Assign_noCCNumber">
        <copy>
          <from expression="string('CreditCardCheck - NO CreditCard')"/>
          <to variable="gOrderProcessorFaultVariable"
              part="code"/>
        </copy>
      </assign>
      <throw name="Throw_NoCreditCard">

```

```
        faultVariable="gOrderProcessorFaultVariable"
        faultName="ns9:OrderProcessingFault"/>
    </sequence>
</catch>
<catch faultName="ns2:InvalidCredit">
    <sequence>
        <assign name="Assign_InvalidCreditFault">
            <copy>
                <from expression="concat(bpws:getVariableData
                    ('gOrderInfoVariable','/ns4:orderInfoVOSDO/ns4:CardTypeCode'), '
                    is not a valid creditcard type')"/>
                <to variable="gOrderProcessorFaultVariable"
                    part="summary"/>
            </copy>
            <copy>
                <from expression="string('CreditCardCheck - NOT VALID')"/>
                <to variable="gOrderProcessorFaultVariable"
                    part="code"/>
            </copy>
        </assign>
        <throw name="Throw_OrderProcessingFault"
            faultName="ns9:OrderProcessingFault"
            faultVariable="gOrderProcessorFaultVariable"/>
    </sequence>
</catch>
</faultHandlers>
```

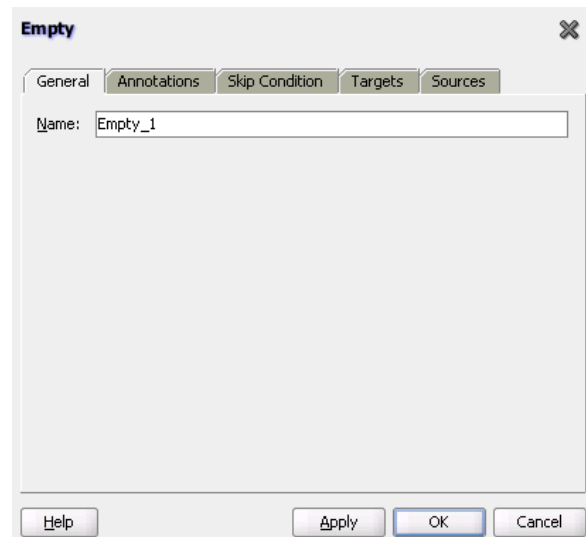
11.10.8 How to Create an Empty Activity to Insert No-Op Instructions into a Business Process

There is often a need to use an activity that does nothing. An example is when a fault must be caught and suppressed. In this case, you can use the empty activity to insert a no-op instruction into a business process.

To create an empty activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag an **Empty** activity into the designer.
3. Double-click the **Empty** activity.

The Empty dialog appears, as shown in [Figure 11-12](#).

Figure 11-12 Empty Activity

4. Optionally enter a name.
5. Click OK.

11.10.9 What Happens When You Create an Empty Activity

The syntax for an `empty` activity is shown in [Example 11-27](#).

Example 11-27 Empty Activity

```
<empty standard-attributes>
  standard-elements
</empty>
```

If no `catch` or `catchAll` is selected, the fault is not caught by the current scope and is rethrown to the immediately enclosing scope. If the fault occurs in (or is rethrown to) the global process scope, and there is no matching fault handler for the fault at the global level, the process terminates abnormally. This is as though a terminate activity (described in [Section 11.13.1, "Stopping a Business Process Instance with the Terminate Activity in BPEL 1.1"](#)) had been performed.

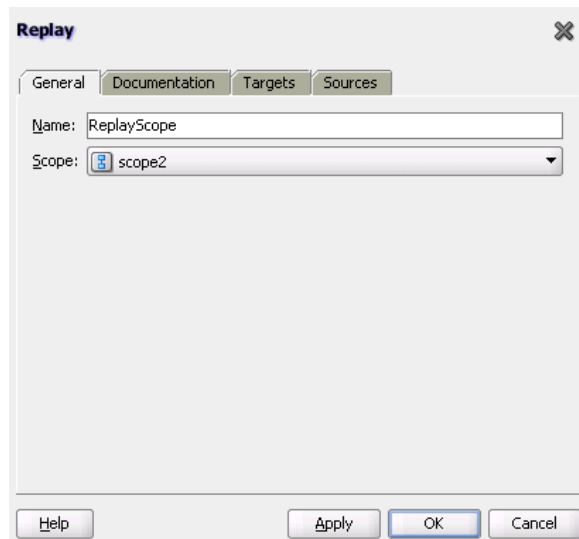
11.11 Re-executing Activities in a Scope Activity with the Replay Activity

You can create a replay activity inside a scope activity to re-execute all of the activities inside the scope.

11.11.1 How to Create a Replay Activity

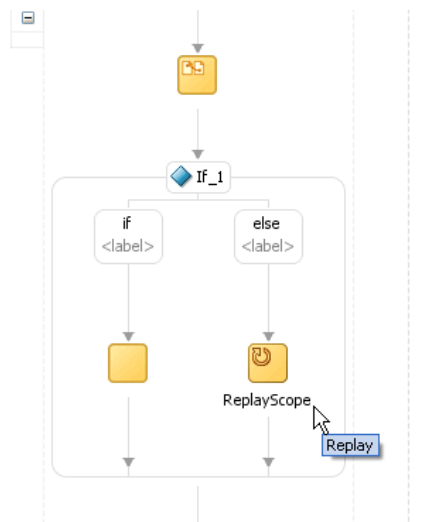
To create a replay activity:

1. In the Component Palette, expand **Oracle Extensions**.
2. Drag a **Replay** activity into the designer.
3. Double-click the **Replay** activity.
4. Enter an optional name.
5. Select the scope to re-execute, as shown in [Figure 11-13](#).

Figure 11–13 *Replay Dialog*

6. Click **Apply**, then click **OK**.
7. Continue with the design of your scope activity.

When complete, design of the scope activity can look similar to that shown in [Figure 11–14](#).

Figure 11–14 *Replay Activity in a Scope Activity*

11.11.2 What Happens When You Create a Replay Activity

[Example 11–28](#) shows the `.bpel` file after design is complete for a replay activity in a BPEL project that supports BPEL version 2.0. In BPEL 2.0, the replay activity is wrapped in an `extensionActivity` element.

Example 11–28 *Replay Activity*

```
<scope name="scope2">
  <sequence>
    <assign>
```

```

    <copy>
      <from>$counter2 + 1</from>
      <to>$counter2</to>
    </copy>
  </assign>
  <scope name="scope3">
    <sequence>
      <assign>
        <copy>
          <from>$counter + 1</from>
          <to>$counter</to>
        </copy>
      </assign>
      <if>
        <condition>$counter = 3</condition>
        <empty/>
      <else>
        <extensionActivity>
          <bpelx:replay name="ReplayScope" scope="scope2" />
        </extensionActivity>
      </else>
    </if>
  </sequence>
</scope>
</sequence>
</scope>

```

In BPEL 1.1, the replay activity is coded as a `bpelx` extension.

```
<bpelx:replay name="ReplayScope" scope="Scope2" />
```

11.12 Using Compensation After Undoing a Series of Operations

Compensation occurs when the BPEL process service component cannot complete a series of operations after some have completed, and the BPEL process service component must backtrack and undo the previously completed transactions. For example, if a BPEL process service component is designed to book a rental car, a hotel, and a flight, it may book the car and the hotel and then be unable to book a flight for the right day. In this case, the BPEL flow performs compensation by going back and unbooking the car and the hotel.

In a scope activity, the compensation handler can reverse previously completed process steps. The compensation handler can be invoked after successful completion of its associated scope with either of the following activities.

- **Compensate activity** (in BPEL version 1.1 and 2.0 projects)
This activity causes the compensation handler of all successfully completed and not yet compensated child scopes to be executed in default order.
- **compensateScope activity** (in a BPEL version 2.0 project)
This activity causes the compensation handler of one specific successfully completed scope to be executed.

11.12.1 Using a Compensate Activity

You can invoke a compensation handler by using the `compensate` activity, which names the scope for which the compensation is to be performed (that is, the scope whose compensation handler is to be invoked). A compensation handler for a scope is

available for invocation only when the scope completes normally. Invoking a compensation handler that has not been installed is equivalent to using the empty activity (it is a no-op). This ensures that fault handlers do not have to rely on state to determine which nested scopes have completed successfully. The semantics of a process in which an installed compensation handler is invoked multiple times are undefined.

The ability to explicitly invoke the compensate activity is the underpinning of the application-controlled error-handling framework of *Business Process Execution Language for Web Services Specification*. You can use this activity only in the following parts of a business process:

- In a fault handler of the scope that immediately encloses the scope for which compensation is to be performed.
- In the compensation handler of the scope that immediately encloses the scope for which compensation is to be performed.

For example:

```
<compensate scope="RecordPayment" />
```

If a scope being compensated by name was nested in a loop, the BPEL process service component invokes the instances of the compensation handlers in the successive iterations in reverse order.

If the compensation handler for a scope is absent, the default compensation handler invokes the compensation handlers for the immediately enclosed scopes in the reverse order of the completion of those scopes.

The compensate form, in which the scope name is omitted in a compensate activity, explicitly invokes this default behavior. This is useful when an enclosing fault or compensation handler must perform additional work, such as updating variables or sending external notifications, in addition to performing default compensation for inner scopes. The compensate activity in a fault or compensation handler attached to the outer scope invokes the default order of compensation handlers for completed scopes directly nested within the outer scope. You can mix this activity with any other user-specified behavior except for the explicit invocation of the nested scope within the outer scope. Explicitly invoking compensation for such a scope nested within the outer scope disables the availability of default-order compensation.

11.12.2 How to Create a Compensate Activity

To create a compensate activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag a **Compensate** activity into the designer
3. Double-click the **Compensate** activity.
4. Select a scope activity in which to invoke the compensation handler, as shown in [Figure 11-15](#).

Figure 11–15 Compensate Activity

The screenshot shows a 'Compensate' dialog box with the following details:

- General Tab:**
 - Name:** CompensateCR
 - Scope:** assignCR
- Buttons:** Help, Apply, OK, Cancel

5. Click **Apply**, then **OK**.

11.12.3 What Happens When You Create a compensate Activity

If an invoke activity has a compensation handler defined inline, then the name of the activity is the name of the scope to be used in the compensate activity. The syntax is shown in [Example 11–29](#):

Example 11–29 Compensation Handler

```
<compensate scope="ncname"? standard-attributes>
  standard-elements
</compensate>
```

11.12.4 Using a compensateScope Activity in BPEL 2.0

The compensateScope activity is used to start compensation on a specified inner scope that has already completed successfully. This activity must only be used from within a fault handler, another compensation handler, or a termination handler.

When you create a compensateScope activity, you select a target that must refer to the immediately-enclosed scope. The scope must include a fault handler or compensation handler.

11.12.5 How to Create a compensateScope Activity

Note: This activity is supported in BPEL 2.0 projects.

To create a compensateScope activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag a **CompensateScope** activity into the designer
3. Double-click the **CompensateScope** activity.

4. Select a specific scope activity in which to invoke the compensation handler.
Figure 11–16 provides details.

Figure 11–16 *CompensateScope Activity*

5. Click **Apply**, then **OK**.

11.12.6 What Happens When You Create a compensateScope Activity

Example 11–30 shows the .bpel file after design is complete for a compensateScope activity. The compensateScope activity is defined in a catchall fault handler. The scope in which to invoke the compensation handler is defined.

Example 11–30 *compensateScope Activity*

```
<scope name="ScopeAssignCreditRating">
  <faultHandlers>
    <catchAll>
      <compensateScope target="ScopeAssignScreditRating2" />
    </catchAll>
  </faultHandlers>
  <sequence>
    <scope name="ScopeAssignScreditRating2">
      <compensationHandler>
        <!-- undo work -->
      </compensationHandler>
      <!-- do some work -->
    </scope>
    <!-- do more work -->
    <!-- a fault is thrown here; results of ScopeAssignScreditRating2 must be
undone -->
  </sequence>
</scope>
```

11.13 Stopping a Business Process Instance

You can stop a business process instance with either of the following activities:

- Terminate activity (in a BPEL version 1.1 project)

- Exit activity (in a BPEL version 2.0 project)

11.13.1 Stopping a Business Process Instance with the Terminate Activity in BPEL 1.1

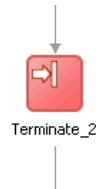
The terminate activity immediately terminates the behavior of a business process instance within which the terminate activity is performed. All currently running activities must be terminated as soon as possible without any fault handling or compensation behavior. The terminate activity does not send 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.

11.13.1.1 How to Create a Terminate Activity

To create a terminate activity:

1. In the Component Palette in Oracle JDeveloper, expand **BPEL Constructs**.
2. Drag a **Terminate** activity into the designer. [Figure 11-17](#) provides an example.

Figure 11-17 Terminate Activity



3. Double-click the **terminate** activity.
4. Optionally enter a name.
5. Click **OK**.

11.13.1.2 What Happens When You Create a Terminate Activity

The syntax for the `terminate` activity is shown in [Example 11-31](#). This stops the business process instance.

Example 11-31 Terminate Activity

```
<terminate standard-attributes>
  standard-elements
</terminate>
```

11.13.2 Immediately Ending a Business Process Instance with the Exit Activity in BPEL 2.0

You can use the exit activity to immediately end all currently running activities on all parallel branches without involving any termination handling, fault handling, or compensation handling mechanisms. This activity is useful for environments in which there may not be a reasonable way for dealing with unexpected, severe failures.

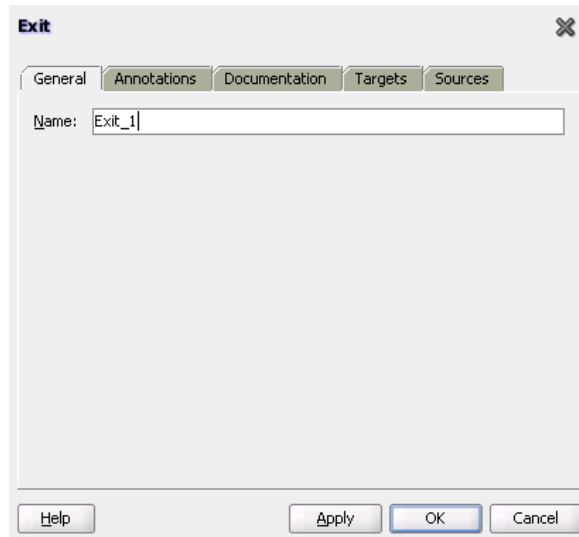
Note: Any open conversations are also impacted by the exit activity. For example, other partners interacting with the process may wait for a response that never arrives.

11.13.2.1 How to Create an Exit Activity

To create an exit activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag an **Exit** activity into the section of your BPEL process in which you want to execute the exit activity.
3. Double-click the **Exit** activity, as shown in [Figure 11-18](#).

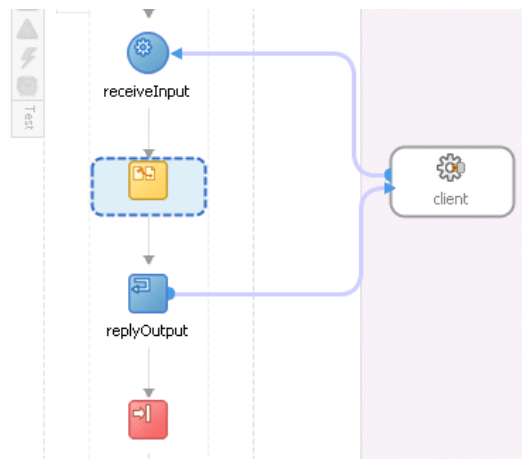
Figure 11-18 Exit Activity



4. Optionally enter a name.
5. Click **Apply**, then **OK**.

When complete, the exit activity in a BPEL process appears similar to that shown in [Figure 11-19](#).

Figure 11-19 Exit Activity in a BPEL Process



11.13.2.2 What Happens When You Create an Exit Activity

[Example 11-32](#) shows the `.bpe1` file after design is complete for an exit activity.

Example 11–32 Exit Activity

```

<sequence>
  <!-- receive input from requestor -->
  <receive name="receiveInput" partnerLink="client" portType="tns:Test"
    operation="process" variable="input" createInstance="yes"/>
  <assign>
    <copy>
      <from>$input.payload</from>
      <to>$output.payload</to>
    </copy>
  </assign>
  <!-- respond output to requestor -->
  <reply name="replyOutput" partnerLink="client"
    portType="tns:Test" operation="process" variable="output"/>
  <exit/>
</sequence>

```

11.14 Throwing Faults with Assertion Conditions

You can specify an assertion condition that is executed upon receipt of a callback message in request-response invoke activities, receive activities, reply activities, and onMessage branches of pick and scope activities. The assertion specifies an XPath expression that, when evaluated to false, causes a BPEL fault to be thrown from the activity. This condition provides an alternative to creating a potentially large number of switch, assign, and throw activities after a partner callback.

Note: The assertion condition is only available in BPEL projects that support BPEL version 1.1

The assertion condition is specified as a nested extension element. [Example 11–33](#) provides details.

Example 11–33 Assertion Condition

```

<invoke | receive | onMessage>
  standard-elements
  <bpelx:postAssert name="ncname"? expression="boolean-expr" faultName="QName"+
    message="generic-expr"/> *
</invoke | receive | onMessage>

```

The `bpelx:postAssert` extension specifies the XPath expression to evaluate upon receipt of a callback message from a partner. If the assertion expression returns a false boolean value, the specified fault is thrown from the activity. If the assertion expression returns a true boolean value, no fault is thrown and the activities following the invoke activity, receive activity, or the onMessage branch of pick and scope activities are executed as in a normal BPEL process flow.

The `bpelx:preAssert` or `bpelx:postAssert` extension is similar to the Java `assert` statement. In Java, if the `assert` expression does not evaluate to true, an error is reported by the JVM. Similarly, the expression in the `bpelx:preAssert` or `bpelx:postAssert` extension must evaluate to true; otherwise, the specified fault is thrown.

For example, with the invoke activity shown in [Example 11–34](#), if the XPath expression specified in the assertion condition returns false, the `NegativeCredit` fault is thrown.

Example 11–34 Invoke Activity

```

<scope>
  <faultHandlers>
    <catch faultName="services:NegativeCredit" faultVariable="crError">
      <empty/>
    </catch>
  </faultHandlers>
  <sequence>
    <invoke name="invokeCR" partnerLink="creditRatingService"
      portType="services:CreditRatingService" operation="process"
      inputVariable="crInput" outputVariable="crOutput">
      <bpelx:postAssert name="negativeCredit"
        expression="$crOutput.payload/tns:rating > 0"
        faultName="services:NegativeCredit" message="'Negative
        Credit'" />
      </invoke>
    </sequence>
  </scope>

```

The optional name attribute for `bpelx:preAssert` or `bpelx:postAssert` is used while creating the audit trail event message. The name in this instance enables you to identify the assertion element in case multiple assertions are specified. If no name attribute is specified, the line number of the assertion element in the BPEL file may be used.

11.14.1 `bpelx:postAssert` and `bpelx:preAssert` Extensions

Depending upon the activity, you can specify when to execute a condition by clicking the **Add** icon in the **Assertions** tab of invoke, receive, reply, and onMessage branches of pick and scope activities, and selecting either **Pre Assert** or **Post Assert**. Based on your selection, the following `bpelx` extensions are used:

- `bpelx:preAssert`: If you select **Pre Assert**, the condition is executed before the invoke or reply activity send out the outbound message.
- `bpelx:postAssert`: If you select **Post Assert**, the condition is executed after an invoke activity, receive activity, or onMessage branch receives the inbound message.

[Example 11–35](#) shows multiple `bpelx:postAssert` extensions in a receive activity:

Example 11–35 `bpelx:postAssert` Extension in a Receive Activity

```

<receive name="Receive_1" createInstance="no"
  variable="Receive_1_processResponse_InputVariable"
  partnerLink="AsyncBPELService"
  portType="ns1:AsyncBPELServiceCallback"
  bpelx:for="'PT10S'"
  operation="processResponse">
  <bpelx:postAssert name="assert1" expression="true()" message="'assert
  true failed'" faultName="client:fault1"/>
  <bpelx:postAssert name="assert2" expression="false()" message="'assert
  false failed'" faultName="client:fault2"/>
</receive>

```

[Example 11–36](#) shows multiple `bpelx:preAssert` extensions in an invoke activity:

Example 11–36 `bpelx:preAssert` Extension in a Invoke Activity

```

<invoke name="Invoke_1" inputVariable="Invoke_1_process_InputVariable"

```

```

        outputVariable="Receive_1_processResponse_InputVariable"
        partnerLink="SyncBPELService" portType="ns1:SyncBPELService"
        operation="process">
        <bpelx:preAssert name="assert1" expression="true()" message="'assert true
failed'"/>
        <bpelx:preAssert name="assert2"
expression="bpws:getVariableData('counter') = 3" message="concat('The value of
counter is ', $counter)"/>

```

For information on using the Assertions tab, see [Section 11.14.8, "How to Create Assertion Conditions."](#)

11.14.2 Use of faultName and message Attributes

You can specify the `faultName` and `message` attributes of the `bpelx:postAssert` element, as shown in [Example 11-37](#).

Example 11-37 *faultName and message Attributes*

```

<invoke | receive | onMessage>
    standard-elements
    <bpelx:postAssert name="ncname"? expression="boolean-expr" faultName="QName"+
message="generic-expr"/> *
</invoke | receive | onMessage>

```

If you do not specify the `faultName` attribute, the fault defaults to `bpelx:postAssertFailure`. If the `message` attribute is not specified, the `message` value defaults to the name of the activity.

```
<bpelx:postAssert expression="boolean-expr" />
```

The specified fault is thrown whenever the assertion condition evaluates to false. Analysis is performed on the `faultName` `QName` to ensure that it properly resolves to a fault that has been defined in the partner WSDL `portType`. The `message` expression is a general expression that can evaluate to any XPath value type (string, number, or boolean). If a nonstring value is returned, the string equivalent of the value is used.

11.14.3 Multiple Assertions

You can nest multiple assertions in receive activities, invoke activities, and the `onMessage` branch of pick and scope activities, with evaluation of the assertions continuing in the order in which they were declared until an expression evaluates to false. [Example 11-38](#) provides details.

Example 11-38 *Nesting Multiple Assertions*

```

<invoke name="invokeCR" partnerLink="creditRatingService"
    portType="services:CreditRatingService" operation="process"
    inputVariable="crInput" outputVariable="crOutput">
    <bpelx:postAssert name="negativeCredit"
expression="$crOutput.payload/tns:rating >
0"
        faultName="services:NegativeCredit" message="'Negative Credit'"
    />
    <bpelx:postAssert name="insufficientCredit"
expression="$crOutput.payload/tns:rating > 600"
        faultName="services:InsufficientCredit" message="'Insufficient
Credit'" />
</invoke>

```

In [Example 11–38](#), the assertion with the expression that checks that the response credit rating is greater than zero is evaluated first. [Table 11–4](#) describes the assertion behavior.

Table 11–4 Assertion Behavior

If The Credit Rating For The Returned Response Is...	Then...
Less than zero	The <code>services:NegativeCredit</code> fault is thrown.
Greater than or equal to zero	The assertion is correct and the second assertion is evaluated.
Less than 600	The <code>services:InsufficientCredit</code> fault is thrown.
Greater than or equal to 600	The assertion is correct and no fault is thrown from the invoke activity.

Any number of assertions can be nested. For no fault to be thrown from the activity, all assertions specified must evaluate to true.

This construct enables you to apply multiple levels of validation on an incoming payload, similar to `if...else if...else` statements in Java.

To enable a fault to always be thrown regardless of validation logic, the assertion expression can be specified as `false()`. This is similar to the `else` construct in Java.

11.14.4 Use of Built-in and Custom XPath Functions and \$variable References

You can also use built-in and custom XPath functions and `$variable` references within the assertion condition. [Example 11–39](#) provides several examples.

Example 11–39 Built-in and Custom XPath Functions

```
<bpelx:postAssert expression="bpws:getVariableData( 'crOutput', 'payload',
  '/tns:rating' ) > 0" ... />

<bpelx:postAssert expression="custom:validateRating()" ... />

<bpelx:postAssert xmlns:fn='http://www.w3.org/2005/xpath-functions'
  expression="fn:false()" ... />
```

If an error is thrown by the XPath expression evaluation, the error is wrapped with a BPEL fault and thrown from the activity.

Faults that are thrown from a request-response invoke activity, receive activity, or onMessage branch of a pick or scope activity because of a failed assertion evaluation can be caught and handled by BPEL's fault policy framework. The fault policy framework enables you to specify the action to take whenever a fault (business or system) is thrown from an invoke activity. For example:

- Retry of the invocation with exponential backoff
- Execution of custom Java classes
- Replay of the immediate scope containing the invoke activity
- Review of the activity by an administrator and the permitting of manual editing of variables

Faults that are not caught and handled within a BPEL process flow are thrown from a BPEL component if the component WSDL declares the fault on the operation. If the

fault is not declared on the operation, the fault is converted into a `FabricInvocationException`, which is a runtime fault. This fault can be caught by any caller components (including BPEL components), but the fault type is no longer the one originally thrown (however, the fault message string still retains traces of the original fault message).

For more information about fault policies, see [Section 11.4, "Using the Fault Management Framework."](#)

11.14.5 Assertion Condition Evaluation Logging of Events to the Instance Audit Trail

Each assertion condition that is evaluated causes an event to be logged to the instance audit trail. The event indicates whether the assertion passed or failed (for failure, the fault name and message are printed). The event also includes the name attribute specified in the assertion element; if no name attribute is provided, the line number of the assertion element in the BPEL process flow is used. The assertion condition printed in the audit event helps identify the assertion and better enables debugging of the flow.

11.14.6 Expressions Not Evaluating to an XML Schema Boolean Type Throw a Fault

If the assertion condition XPath expression does not evaluate to an XML schema boolean type, a `bpelx:postAssertFailure` fault is thrown from the activity. An event in the instance audit trail is also logged indicating the error. [Example 11-40](#) provides details.

Example 11-40 Throwing a `bpelx:assertFailure` Fault

```
<bpelx:postAssert expression="bpws:getVariableData( 'crOutput', 'payload',
'/tns:rating' ) > 0" ... />

<bpelx:postAssert expression="custom:validateRating()" ... />

<bpelx:postAssert xmlns:fn='http://www.w3.org/2005/xpath-functions'
expression="fn:false()" ... />
```

Analysis of the assertion expression is performed by the BPEL compiler and errors are reported if an expression does not evaluate to an XML schema boolean type. For custom XPath functions, this type of analysis is not performed.

11.14.7 Assertion Conditions in a Standalone Assert Activity

You can also create assertion conditions in a standalone assert activity in BPEL 1.1. The assertion specifies an XPath expression that, when evaluated to false, causes a BPEL fault to be thrown from the activity.

The `bpelx:assert` extension implements assertions in the standalone assert activity:

```
<bpelx:assert name="Assert1" expression="string" message="string"/>
```

For information on using the standalone assert activity, see [Section 11.14.8, "How to Create Assertion Conditions."](#)

11.14.8 How to Create Assertion Conditions

You can create assertion conditions in the following activities:

- In invoke activities, receive activities, reply activities, and OnMessage branches

- In standalone assert activities

To create assertion conditions in invoke activities, receive activities, reply activities, and OnMessage branches:

1. In the SOA Composite Editor, double-click the version 1.1 BPEL process service component.
2. In the Component Palette, expand **BPEL Constructs**.
3. Drag a **Receive** activity, **Invoke** activity, **Pick** activity, or **Scope** activity into the designer.
4. Expand the **Receive**, **Invoke**, or **onMessage** branch of the **Pick** or **Scope** activity.
5. Click the **Assertions** tab.
6. Click the **Add** icon.
7. Select when to execute the condition.

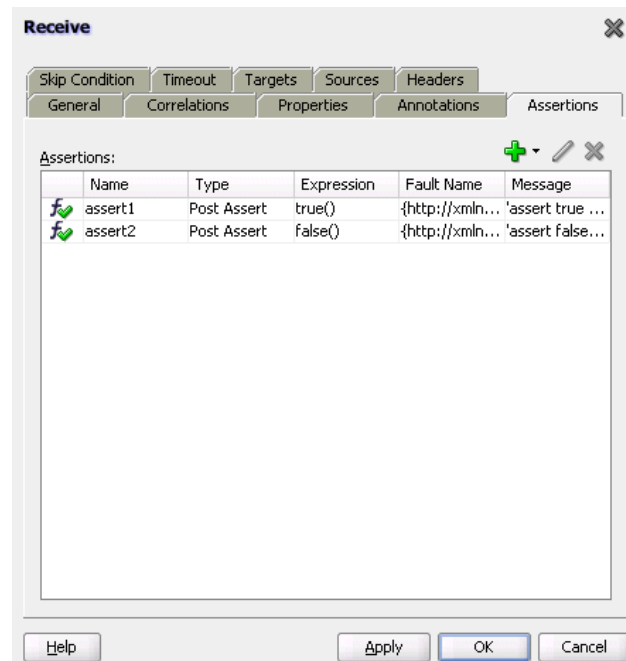
- **Pre Assert:** If selected, the condition is executed before the invoke or reply activity send out the outbound message.
- **Post Assert:** If selected, the condition is executed after an invoke activity, receive activity, or onMessage branch receives the inbound message.

Based on your selection, the Pre Assert or Post Assert dialog is displayed.

8. Specify values for the assertion condition, as shown in [Figure 11–20](#). For this example, Post Assert was selected for an assertion condition on a receive activity.
 - a. Select the **Fault QName** to be thrown by clicking the **Search** icon and selecting an existing fault from the Fault Chooser dialog. You can also provide your own values for the **Namespace URI** and **Local Part** fields of the fault. If you do not specify anything for the **Fault QName**, then a `bpelx:assertFailure` fault is thrown.

Figure 11–20 Assertion Condition Values

9. When complete, click **OK** to return to the **Assertions** tab of the activity. The completed assertion condition is displayed, as shown in [Figure 11–21](#).

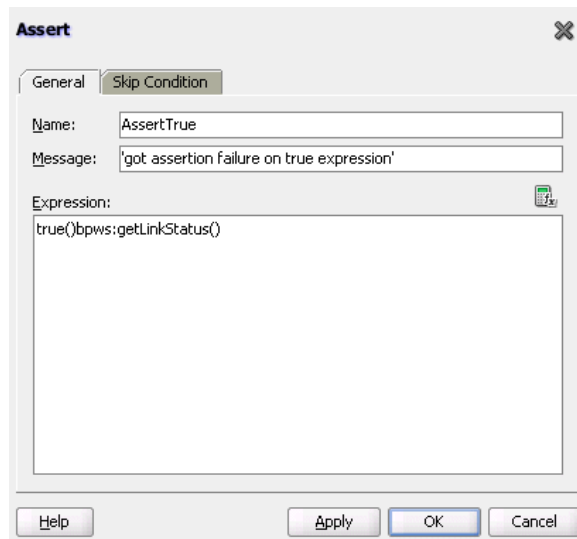
Figure 11–21 Assertions Tab with Data

10. Click **Apply**, then **OK**.

To create an assertion condition in standalone assert activities:

1. In the SOA Composite Editor, double-click the version 1.1 BPEL process service component.
2. In the Component Palette, expand **BPEL Constructs**.
3. Drag an **Assert** activity into the designer.
4. Expand the **Assert** activity.
5. To the right of the **Expression** field, click the **XPath Expression Builder** icon.
6. Create an expression.
7. When complete, click **OK**.

The Assert dialog looks as shown in [Figure 11–22](#).

Figure 11–22 Assert Dialog

8. Click **Apply**, then **OK**.

11.14.9 How to Disable Assertions

You can disable assertions in either of two ways:

- By setting the System MBean Browser property **DisableAsserts** to **true** in Oracle Enterprise Manager Fusion Middleware Control.
- By setting `bpel.config.disableAsserts` to `true` in the `composite.xml` file of the SOA composite application, as shown in [Example 11–41](#).

Example 11–41 Disable Assertions

```
<component name="AsyncBPELClient">
  <implementation.bpel src="AsyncBPELClient.bpel"/>
  <property name="bpel.config.disableAsserts">true</property>
</component>
```

For more information about setting System MBean Browser properties, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

11.14.10 What Happens When You Create Assertion Conditions

The code segment in the `.bpel` file defines the specific operation after design completion.

For [Example 11–42](#), the `bpelx:assert` condition in the `invoke` activity, when evaluated to false (for example, a credit rating of 0 is submitted), returns a `Negative Credit` message. If the condition evaluates to true, no fault is thrown from the `invoke` activity and the remaining activities in the BPEL process flow are executed normally.

Example 11–42 Assertion Condition in an Invoke Activity

```
<invoke name="callbackClient" partnerLink="internalwarehouseservice_client"
  portType="client:InternalWarehouseServiceCallback" operation="processResponse"
  inputVariable="outputVariable">
  <bpelx:assert name="negativeCredit"
    expression="$scrOutput.payload/tns:rating > 0"
```



```
message="Negative Credit"/>  
</invoke>
```

In [Example 11–43](#), the `bpelx:assert` condition in the standalone assert activity, when evaluated to false, returns a got assertion failure on true expression message. If the condition evaluates to true, no fault is thrown from the assert activity and the remaining activities in the BPEL process flow are executed normally.

Example 11–43 Assertion Condition in a Standalone Assert Activity

```
<bpelx:assert expression="true()bpws:getLinkStatus()" message="'got assertion  
failure on true expression'"
```

Transaction and Fault Propagation Semantics in BPEL Processes

This chapter describes transaction and fault propagation semantics in Oracle BPEL Process Manager.

This chapter includes the following sections:

- [Section 12.1, "Introduction to Transaction Semantics"](#)
- [Section 12.2, "Introduction to Execution of One-way Invocations"](#)

12.1 Introduction to Transaction Semantics

Transaction semantics in release 11g enable you to use the underlying Java Transaction API (JTA) infrastructure used in the execution of components. This section describes transaction semantics for Oracle BPEL Process Manager

12.1.1 Oracle BPEL Process Manager Transaction Semantics

As with previous releases, Oracle BPEL Process Manager by default creates a new transaction on a request basis. That is, if a transaction exists, it is suspended, and a new transaction is created. Upon completion of the child (new) transaction, the master (suspended) transaction resumes.

However, if the request is asynchronous (that is, one-way), the transaction is either:

- Inherited for insertion into the dehydration store (table `dlv_message`).
- Enlisted transparently into the transaction (if one exists).

There is no message loss. Either the invocation message is inserted into the dehydration store for processing or the consumer is notified through a fault.

In release 10.1.3.x, there were several properties to set on the consuming process (that is, on the partner link) and the providing process. This enabled you to chain an execution into a single global transaction. On the consuming side, you set `transaction=participate` on the partner link binding in the `bpel.xml` file. On the providing side, you set `transaction=participate` in the `<configurations>` section of `bpel.xml`.

In release 11g, you only must set a new `transaction` property on the BPEL component being called (known as the callee process). You add `bpel.config.transaction` into a BPEL process service component section in the `composite.xml` file (note the required prefix of `bpel.config.`). This property configures the transaction behavior for BPEL instances with initiating calls.

[Example 12–1](#) provides details.

Example 12–1 Setting a New Transaction

```
<component name="InternalWarehouseService">
  <implementation.bpel src="InternalWarehouseService.bpel" />
  <property name="bpel.config.transaction"
    many="false" type="xs:string">required | requiresNew</property>
</component>
```

There are two possible values: `required` and `requiresNew`. [Table 12–1](#) describes these values and summarizes the behavior of the BPEL instance based on the settings.

Table 12–1 *bpel.config.transaction Property Behavior*

For...	With <code>bpel.config.transaction</code> Set to <code>required</code> ...	With <code>bpel.config.transaction</code> Set to <code>requiresNew</code> ...
Request/response (initiating) invocations	The caller's transaction is joined (if there is one) or a new transaction is created (if there is not one).	A new transaction is always created and an existing transaction (if there is one) is suspended.
One-way initiating invocations in which <code>bpel.config.oneWayDeliveryPolicy</code> is set to <code>sync</code> .	Invoked messages are processed using the same thread in the same transaction.	A new transaction is always created and an existing transaction (if there is one) is suspended.

Note: The `bpel.config.transaction` property does not apply for midprocess receive activities. In those cases, another thread in another transaction is used to process the message. This is because correlation is needed and it is always done asynchronously.

For additional information about setting the `bpel.config.transaction` property, see [Section C.1.1, "How to Define Deployment Descriptor Properties."](#)

The following sections describe the transaction and fault behavior of setting `bpel.config.transaction` to either `required` or `requiresNew`.

12.1.1.1 BPELCaller Calls BPELCallee That Has `bpel.config.transaction` Set to `requiresNew`

In [Table 12–2](#), the BPELCaller process calls the BPELCallee process. The BPELCallee process has the property `bpel.config.transaction` set to `requiresNew`. [Table 12–2](#) describes fault propagation and transaction behavior when `bpel.config.transaction` is set to this value.

Table 12–2 *BPELCaller Calls BPELCallee That Has `bpel.config.transaction` Set to `requiresNew`*

If The BPELCallee...	Then The BPELCallee Transaction...	And The BPELCaller...
Replies with a fault (that is, it uses <code><reply></code>).	Is saved.	Gets the fault and can catch it.
Throws a fault that is not handled (that is, it uses <code><throw></code>).	Is rolled back.	Gets the fault and can catch it.

Table 12–2 (Cont.) BPELCaller Calls BPELCallee That Has `bpel.config.transaction` Set to `requiresNew`

If The BPELCallee...	Then The BPELCallee Transaction...	And The BPELCaller...
Replies back with a fault (FaultOne), and then throws a fault (FaultTwo).	Is rolled back.	Gets FaultTwo.
Throws a <code>bpelx:rollback</code> fault (that is, it uses <code><throw></code>).	Is rolled back.	Gets a remote fault.

12.1.1.2 BPELCaller Calls BPELCallee That Has `bpel.config.transaction` Set to `required`

In [Table 12–3](#), the BPELCaller process calls the BPELCallee process. The BPELCallee process has the property `bpel.config.transaction` set to `required`. [Table 12–3](#) describes fault propagation and transaction behavior when `bpel.config.transaction` is set to this value.

Table 12–3 BPELCaller Calls BPELCallee That Has `bpel.config.transaction` Set to `required`

If The BPELCallee...	Then The BPELCaller...
Replies with a fault (that is, it uses <code><reply></code>).	Gets the fault and can catch it. The BPELCaller owns the transaction. Therefore, if it catches it, the transaction is committed. If the BPELCaller does not handle it, a global rollback occurs.
Throws a fault (that is, it uses <code><throw></code>).	Gets the fault and can catch it.
Replies back with a fault (FaultOne), and then throws a fault (FaultTwo).	Gets FaultTwo.
Throws (that is, it uses <code><throw></code>) a <code>bpelx:rollback</code> fault.	Gets its transaction rolled back; there is no way to catch it. This fault cannot be handled.

As an example, assume you create two synchronous processes (BPELMaster and BPELChild) that each use the same database adapter reference to insert the same record (and therefore, causes a permission key (PK) violation). The `xADatasourceName` is set for both.

Without `bpel.config.transaction` set, after the fault occurs, and it is not handled, BPELChild is rolled back. If BPELMaster has a catch block, its transaction is committed. Therefore, you end up with the record from BPELMaster in the database.

If you do not catch the fault in BPELMaster as well, you get a second rollback (however, in two different transactions).

If `bpel.config.transaction` is set to `required` for the same test case and no fault handlers are in place, the entire transaction is rolled back based on BPELMaster's unhandled fault.

If you add a fault handler in BPELMaster to catch the fault from BPELChild and throw a rollback fault, the transaction is globally rolled back.

This feature enables you to control transaction boundaries and model end-to-end transactional flows (if your sources and targets are also transactional).

12.2 Introduction to Execution of One-way Invocations

A one-way invocation (with a possible callback) is typically exposed in a WSDL as shown in [Example 12-2](#).

Example 12-2 WSDL Exposure

```
<wsdl:operation name="process">
  <wsdl:input message="client:OrderProcessorRequestMessage" />
</wsdl:operation>
```

This causes the BPEL process service engine to split the execution into two parts:

- For the first part, and always inside the caller transaction, the insertion into the `dlv_message` table of the dehydration store occurs (in release 10.1.3.x, it was inserted into the `inv_message` table).
- For the second part, the transaction and the new thread executes the work items, and a new instance is created.

This has several advantages in terms of scalability, because the service engine's thread pool (invoker threads) executes when a thread is available. However, the disadvantage is that there is no guarantee that it executes immediately.

If you require a synchronous-type call based on a one-way operation, then you can use the `onewayDeliveryPolicy` property, which is similar to the `deliveryPersistPolicy` property of release 10.1.3.x.

Specify `bpel.config.oneWayDeliveryPolicy` in the BPEL process service component section of the `composite.xml` file. If this value is not set in `composite.xml`, the value for `oneWayDeliveryPolicy` in the System MBean Browser in Oracle Enterprise Manager Fusion Middleware Control is used. The following values are possible.

- `async.persist`: Messages are persisted in the database hash map.
- `sync.cache`: Messages are stored in memory.
- `sync`: Direct invocation occurs on the same thread.

For more information about setting the `bpel.config.oneWayDeliveryPolicy` property, see [Section C.1.1, "How to Define Deployment Descriptor Properties."](#)

[Table 12-4](#) describes the behavior when the main process calls the subprocess asynchronously. [Table 12-4](#) is based on the use cases described in [Section 12.1.1.1, "BPELCaller Calls BPELCallee That Has `bpel.config.transaction` Set to `requiresNew`"](#) and [Section 12.1.1.2, "BPELCaller Calls BPELCallee That Has `bpel.config.transaction` Set to `required`."](#)

Table 12-4 Main Process Calls the Subprocess Asynchronously

If...	If The Subprocess Throws Any Fault...	If The Subprocess Throws a <code>bpelx:rollback...</code>
<code>onewayDeliveryPolicy=async.persist</code> (The BPELCallee process runs in a separate thread/transaction.)	The BPELCaller does not get a response because the message is saved in the delivery service. The BPELCallee transaction is rolled back if the fault is not handled.	The BPELCaller does not get a response because the message is saved in the delivery service. The BPELCallee instance is rolled back on the unhandled fault.

Table 12–4 (Cont.) Main Process Calls the Subprocess Asynchronously

If...	If The Subprocess Throws Any Fault...	If The Subprocess Throws a <code>bpelx:rollback</code>...
onewayDeliveryPolicy=sync and transaction=requiresNew (The BPELCallee runs in the same thread, but a different transaction.)	The BPELCaller receives a <code>FabricInvocationException</code> . The BPELCallee transaction rolls back if the fault is not handled.	The BPELCaller receives a <code>FabricInvocationException</code> . The BPELCallee transaction is rolled back.
onewayDeliveryPolicy=sync and transaction=required (The BPELCallee runs in the same thread and the same transaction.)	The BPELCallee faulted. The BPELCaller receives a <code>FabricInvocationException</code> . The BPELCaller has a chance to handle the fault.	The whole transaction is rolled back.

Incorporating Java and Java EE Code in a BPEL Process

This chapter describes how to incorporate sections of Java code into BPEL process service components in SOA composite applications.

This chapter includes the following sections:

- [Section 13.1, "Introduction to Java and Java EE Code in BPEL Processes"](#)
- [Section 13.2, "Incorporating Java and Java EE Code in BPEL Processes"](#)
- [Section 13.3, "Adding Custom Classes and JAR Files"](#)
- [Section 13.4, "Using Java Embedding in a BPEL Process in Oracle JDeveloper"](#)
- [Section 13.5, "Embedding Service Data Objects with `bpelx:exec`"](#)
- [Section 13.6, "Sharing a Custom Implementation of a Class with Oracle BPEL Process Manager"](#)

13.1 Introduction to Java and Java EE Code in BPEL Processes

This chapter explains how to incorporate sections of Java code into a BPEL process. This is particularly useful when there is Enterprise JavaBeans Java code that can perform the necessary function, and you want to use the existing code rather than start over with BPEL.

13.2 Incorporating Java and Java EE Code in BPEL Processes

There are several methods for incorporating Java and Java EE code in BPEL processes:

- Wrap as a Simple Object Access Protocol (SOAP) service
- Embed Java code snippets into a BPEL process with the `bpelx:exec` tag
- Use an XML facade to simplify DOM manipulation
- Use `bpelx:exec` built-in methods
- Use Java code wrapped in a service interface

13.2.1 How to Wrap Java Code as a SOAP Service

You can wrap the Java code as a SOAP service. This method requires that the Java application have a BPEL-compatible interface. A Java application wrapped as a SOAP service appears as any other web service, which can be used by many different kinds of applications. There are also tools available for writing SOAP wrappers.

13.2.2 What You May Need to Know About Wrapping Java Code as a SOAP Service

A Java application wrapped as a SOAP service has the following drawbacks:

- There may be reduced performance due to the nature of converting between Java and SOAP, and back and forth.
- Since SOAP inherently has no support for transactions, this method loses atomic transactionality, that is, the ability to perform several operations in an all-or-none mode (such as debiting one bank account while crediting another, where either both transactions must be completed, or neither of them).

13.2.3 How to Embed Java Code Snippets into a BPEL Process with the `bpelx:exec` Tag

You can embed Java code snippets directly into the BPEL process using the Java BPEL `exec` extension `bpelx:exec`. The benefits of this approach are speed and transactionality. It is recommended that you incorporate only small segments of code. BPEL is about separation of business logic from implementation. If you remove a lot of Java code in your process, you lose that separation. Java embedding is recommended for short utility-like operations, rather than business code. Place the business logic elsewhere and call it from BPEL.

The server executes any snippet of Java code contained within a `bpelx:exec` activity, within its Java Transaction API (JTA) transaction context.

The BPEL tag `bpelx:exec` converts Java exceptions into BPEL faults and then adds them into the BPEL process.

The Java snippet can propagate its JTA transaction to session and entity beans that it calls.

For example, a `SessionBeanSample.bpel` file uses the `bpelx:exec` tag shown in [Example 13-1](#) to embed the `invokeSessionBean` Java bean:

Example 13-1 *bpelx:exec* Extension

```
<bpelx:exec name="invokeSessionBean" language="java" version="1.5">
  <![CDATA[
    try {
      Object homeObj = lookup("ejb/session/CreditRating");
      Class cls = Class.forName(
        "com.otn.samples.sessionbean.CreditRatingServiceHome");
      CreditRatingServiceHome ratingHome = (CreditRatingServiceHome)
        PortableRemoteObject.narrow(homeObj,cls);
      if (ratingHome == null) {
        addAuditTrailEntry("Failed to lookup 'ejb.session.CreditRating'"
          + ". Ensure that the bean has been"
          + " successfully deployed");
        return;
      }
      CreditRatingService ratingService = ratingHome.create();

      // Retrieve ssn from scope
      Element ssn =
        (Element)getVariableData("input","payload","/ssn");

      int rating = ratingService.getRating( ssn.getNodeValue() );
      addAuditTrailEntry("Rating is: " + rating);

      setVariableData("output", "payload",
        "/tns:rating", new Integer(rating));
    }
  ]]>
</bpelx:exec>
```

```

    } catch (NamingException ne) {
        addAuditTrailEntry(ne);
    } catch (ClassNotFoundException cnfe) {
        addAuditTrailEntry(cnfe);
    } catch (CreateException ce) {
        addAuditTrailEntry(ce);
    } catch (RemoteException re) {
        addAuditTrailEntry(re);
    }
  ]]>
</bpelx:exec>

```

13.2.4 How to Embed Java Code Snippets in a BPEL Process in BPEL 2.0

The examples in this chapter focus primarily on how to embed Java code snippets with the `bpelx:exec` extension. For BPEL projects that support version 2.0 of the BPEL specification, the syntax is slightly different. The `bpelx:exec` extension and Java code are wrapped in an `<extensionActivity>` element. [Example 13–2](#) provides details.

Example 13–2 *bpelx:exec Extension in BPEL 2.0*

```

<extensionActivity>
  <bpelx:exec language="java">
    <![CDATA[
      java code
    ]]>
  </bpelx:exec>
</extensionActivity>

```

When you drag a **Java Embedding** activity into a BPEL process in Oracle BPEL Designer, the `<extensionActivity>` element and `bpelx:exec` tag are automatically added.

[Example 13–3](#) shows the import syntax for BPEL 2.0:

Example 13–3 *Import Syntax in BPEL 2.0*

```

<import location="class/package name"
  importType="http://schemas.oracle.com/bpel/extension/java"/>

```

[Example 13–4](#) shows a BPEL file with two Java embedding activities for a project that supports BPEL version 2.0.

Example 13–4 *Java Embedding Activities in a BPEL File for Version 2.0*

```

<process name="Test" targetNamespace="http://samples.otn.com/bpel2.0/ch10.9"
  . . .
  . . .
  <import location="oracle.xml.parser.v2.XMLElement"
    importType="http://schemas.oracle.com/bpel/extension/java"/>
  . . .
  <sequence>
    . . .
    <extensionActivity>
      <bpelx:exec language="java">
        XMLElement elem = (XMLElement) getVariableData("output", "payload");
        elem.setTextContent("set by java exec");
      </bpelx:exec>
    </extensionActivity>
  </sequence>
</process>

```

```

<extensionActivity>
  <bpelx:exec language="java">
    <![CDATA[XMLElement elem = (XMLElement) getVariableData("output",
      "payload");
    String t = elem.getTextContent();
    elem.setTextContent(t + ", set by java exec 2");]]>
  </bpelx:exec>
</extensionActivity>
. . .
</sequence>
</process>

```

For information on using this activity, see [Section 13.4, "Using Java Embedding in a BPEL Process in Oracle JDeveloper."](#)

13.2.5 How to Use an XML Facade to Simplify DOM Manipulation

You can use an XML facade to simplify DOM manipulation. Oracle BPEL Process Manager provides a lightweight Java Architecture for XML Binding (JAXB)-like Java object model on top of XML (called a facade). An XML facade provides a Java bean-like front end for an XML document or element that has a schema. Facade classes can provide easy manipulation of the XML document and element in Java programs.

You add the XML facade by using a `createFacade` method within the `bpelx:exec` statement in the `.bpel` file. [Example 13-5](#) provides an example:

Example 13-5 Addition of XML facade

```

<bpelx:exec name= ...
  <![CDATA
    ...
    Element element = ...
    (Element) getVariableData("input", "payload", "/loanApplication/"):
    //Create an XMLFacade for the Loan Application Document
    LoanApplication xmlLoanApp=
      LoanApplicationFactory.createFacade(element);
    ...
  ]]>

```

13.2.6 How to Use `bpelx:exec` Built-in Methods

[Table 13-1](#) lists a set of `bpelx:exec` built-in methods that you can use to read and update scope variables, instance metadata, and audit trails.

Table 13-1 Built in Methods for `bpelx:exec`

Method Name	Description
<code>Object lookup(String name)</code>	JNDI access
<code>long getInstanceId()</code>	Unique ID associated with each instance
<code>String setTitle(String title) /</code> <code>String getTitle()</code>	Title of this instance
<code>String setStatus(String status) /</code> <code>String getStatus()</code>	Status of this instance
<code>void</code> <code>setCompositeInstanceTitle(String</code> <code>title)</code>	Set the composite instance title

Table 13–1 (Cont.) Built in Methods for *bpelx:exec*

Method Name	Description
<code>void setIndex(int i, String value)</code> <code>/ String getIndex(int i)</code>	Six indexes can be used for a search
<code>void setCreator(String creator)</code> / <code>String getCreator()</code>	Who initiated this instance
<code>void setCustomKey(String customKey</code> <code>) / String getCustomKey()</code>	Second primary key
<code>void setMetadata(String metadata)</code> <code>/ String getMetadata ()</code>	Metadata for generating lists
<code>String getPreference(String key)</code>	Access preference
<code>void addAuditTrailEntry(String</code> <code>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)</code> <code>throws BPELFault</code>	Access and update variables stored in the scope
<code>Object getVariableData(String name,</code> <code>String partOrQuery) throws BPELFault</code>	Access and update variables
<code>Object getVariableData(String name,</code> <code>String part, String query)</code>	Access and update variables
<code>void setVariableData(String name,</code> <code>Object value)</code>	Set variable data
<code>void setVariableData(String name,</code> <code>String part, Object value)</code>	Set variable data
<code>void setVariableData(String name,</code> <code>String part, String query, Object</code> <code>value)</code>	Set variable data

13.2.7 How to Use Java Code Wrapped in a Service Interface

Not all applications expose a service interface. You may have a scenario in which a business process must use custom Java code. For this scenario, you can:

- Write custom Java code.
- Create a service interface in which to embed the code.
- Invoke the Java code as a web service over SOAP.

For example, assume you create a BPEL process service component in a SOA composite application that invokes a service interface through a SOAP reference binding component. For this example, the service interface used is an Oracle Application Development Framework (ADF) Business Component.

The high-level instructions for this scenario are as follows.

To use Java code wrapped in a service interface:

1. Create an Oracle ADF Business Component service in Oracle JDeveloper.
This action generates a WSDL file and XSD file for the service.
2. Create a SOA application that includes a BPEL process service component. Ensure that the BPEL process service component is exposed as a composite service. This

automatically connects the BPEL process to an inbound SOAP service binding component.

3. Import the Oracle ADF Business Component service WSDL into the SOA composite application.
4. Create a web service binding to the Oracle ADF Business Component service interface.
5. Design a BPEL process in which you perform the following tasks:
 - a. Create a partner link for the Oracle ADF Business Component service portType.
 - b. Create an assign activity. For this example, this step copies data (for example, a static XML fragment) into a variable that is passed to the Oracle ADF Business Component service.
 - c. Create an invoke activity and connect to the partner link you created in Step 5a.
6. Connect (wire) the partner link reference to the composite reference binding component. This reference uses a web service binding to enable the Oracle ADF Business Component service to be remotely deployed.
7. Deploy the SOA application.
8. Invoke the SOA application from the Test Web Service page in Oracle Enterprise Manager Fusion Middleware Control.

For more information on creating Oracle ADF Business Components, see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For more information on invoking a SOA composite application, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

13.3 Adding Custom Classes and JAR Files

You can add custom classes and JAR files to a SOA composite application. A SOA extension library for adding extension classes and JARs to a SOA composite application is available in the `$ORACLE_HOME/soa/modules/oracle.soa.ext_11.1.1` directory. For Oracle JDeveloper, custom classes and JARs are added to the `application_name/project/sca-inf/lib` directory.

13.3.1 How to Add Custom Classes and JAR Files

If the classes are used in `bpelx:exec`, you must also add the JARs with the **BpelcClasspath** property in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control.

To add JARs to BpelcClasspath:

1. From the **SOA Infrastructure** menu, select **SOA Administration > BPEL Properties**.
2. At the bottom of the BPEL Service Engine Properties page, click **More BPEL Configuration Properties**.
3. Click **BpelcClasspath**.
4. In the **Value** field, specify the class path.
5. Click **Apply**.

6. Click **Return**.

In addition, ensure that the JARs are loaded by the SOA composite application.

To add custom classes:

1. Copy the classes to the `classes` directory.
2. Restart Oracle WebLogic Server.

To add custom JARs:

1. Copy the JAR files to this directory or its subdirectory.
2. Run `ant`.
3. Restart Oracle WebLogic Server.

13.4 Using Java Embedding in a BPEL Process in Oracle JDeveloper

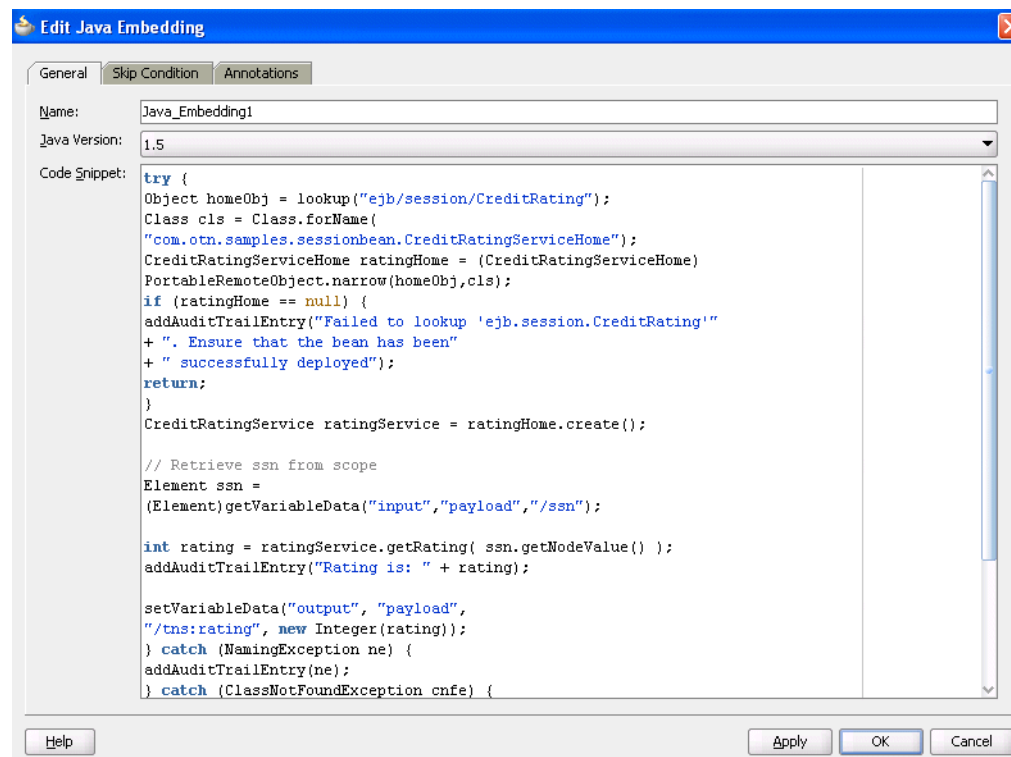
In Oracle JDeveloper, you can add the `bpelx:exec` activity and copy the code snippet into a dialog.

Note: For custom classes, you must include any JAR files required for embedded Java code in the **BpelcClasspath** property in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control. See [Section 13.3.1, "How to Add Custom Classes and JAR Files"](#) for instructions. The JAR files are then added to the class path of the BPEL loader. If multiple JAR files are included, they must be separated by a colon (:) on UNIX and a semicolon (;) on Windows.

13.4.1 How To Use Java Embedding in a BPEL Process in Oracle JDeveloper

To use Java embedding in a BPEL process in Oracle JDeveloper:

1. From the Component Palette, expand **Oracle Extensions**.
2. Drag the **Java Embedding** activity into the designer.
3. Double-click the **Java Embedding** activity to display the Java Embedding dialog.
4. In the **Name** field, enter a name.
5. In the **Code Snippet** field, enter (or cut and paste) the Java code. [Figure 13–1](#) provides details.

Figure 13–1 *bpel:exec Code Example*

Note: As an alternative to writing Java code in the Java Embedding activity, you can place your Java code in a JAR file, put it in the class path, and call your methods from within the Java Embedding activity.

13.4.2 What You May Need to Know About Using `thread.sleep()` in a Java Embedding Activity

If you create and deploy a BPEL process that uses `thread.sleep()` in a Java Embedding activity, the executing thread is blocked and the transaction associated with that thread is prevented from committing. This causes BPEL instances to appear only after the wait is over, which is the expected behavior.

Instead, use a wait activity, which releases the resource upon entering the activity and enables the ongoing transaction to commit and the BPEL instance data to hydrate into the data store.

13.5 Embedding Service Data Objects with `bpelx:exec`

You can embed service data object (SDO) code in the `.bpel` file with the `bpelx:exec` tag. In the syntax provided in [Example 13–6](#), `mytest.apps.SDOHelper` is a Java class that modifies SDOs.

Example 13–6 *Embedding SDO Objects with the `bpelx:exec` tag*

```

</bpelx:exec>
<bpelx:exec name="ModifyInternalSDO" version="1.5" language="java">
    <![CDATA[try{
        Object o = getVariableData("VarSDO");
        Object out = getVariableData("ExtSDO");

```



```

        System.out.println("BPEL:Modify VarSDO... " + o + " ExtSDO: " + out);
    mytest.apps.SDOHelper.print(o);
    mytest.apps.SDOHelper.print(out);
    mytest.apps.SDOHelper.modifySDO(o);
        System.out.println("BPEL:After Modify VarSDO... " + o + " ExtSDO: " + out);
    mytest.apps.SDOHelper.print(o);
    mytest.apps.SDOHelper.print(out);
} catch (Exception e)
{
    e.printStackTrace();
}]]>
</bpelx:exec>

```

Example 13–7 provides an example of the Java classes `modifySDO(o)` and `print(o)` that are embedded in the BPEL file.

Example 13–7 Java Classes

```

public static void modifySDO(Object o){
    if(o instanceof commonj.sdo.DataObject)
    {
        ((DataObject)o).getChangeSummary().beginLogging();
        SDOType type = (SDOType)((DataObject)o).getType();
        HelperContext hCtx = type.getHelperContext();
        List<DataObject> lines =
            (List<DataObject>)((DataObject)o).get("line");
        for (DataObject line: lines) {
            line.set("eligibilityStatus", "Y");
        }
    } else {
        System.out.println("SDOHelper.modifySDO(): " + o + " is not a
            DataObject!");
    }
}
. . .
. . .
public static void print(Object o)    {
    try{
        if(o instanceof commonj.sdo.DataObject)
        {
            DataObject sdo = (commonj.sdo.DataObject)o;
            SDOType type = (SDOType) sdo.getType();
            HelperContext hCtx = type.getHelperContext();
            System.out.println(hCtx.getXMLHelper().save(sdo, type.getURI(),
                type.getName()));
        } else {
            System.out.println("SDOHelper.print(): Not a sdo " + o);
        }
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

13.6 Sharing a Custom Implementation of a Class with Oracle BPEL Process Manager

When you implement a custom Connection Manager class with the same name as a class used by Oracle BPEL Process Manager, you must ensure that the custom class does not override the class used by Oracle BPEL Process Manager.

For example, assume the following is occurring:

- You are using embedded Java in a BPEL project.
- The Connection Manager custom class is overriding the BPEL Connection Manager class.
- A `java.lang.NoClassDefFoundError` is occurring at runtime.

13.6.1 How to Configure the BPEL Connection Manager Class to Take Precedence

To configure the BPEL Connection Manager class to take precedence:

1. Start Oracle JDeveloper.
2. Highlight the BPEL project.
3. From the **Edit** main menu, select **Properties**.
4. Select **Libraries and Classpath**.
5. Click **Add JAR/Directory**.
6. Navigate to the location of the custom JAR file and click **Select**.
This adds the custom Connection Manager JAR file to the classpath.
7. Click **OK**.
8. Redeploy the BPEL project and retest.

Using Events and Timeouts in BPEL Processes

This chapter describes how to use events and timeouts. Because web services can take a long time to return a response, a BPEL process service component must be able to time out and continue with the rest of the flow after a period of time.

This chapter includes the following sections:

- [Section 14.1, "Introduction to Event and Timeout Concepts"](#)
- [Section 14.2, "Creating a Pick Activity to Select Between Continuing a Process or Waiting"](#)
- [Section 14.3, "Setting Timeouts for Request-Response Operations in Receive Activities"](#)
- [Section 14.4, "Creating a Wait Activity to Set an Expiration Time"](#)
- [Section 14.5, "Specifying Events to Wait for Message Arrival with an OnEvent Branch in BPEL 2.0"](#)
- [Section 14.6, "Setting Timeouts for Synchronous Processes"](#)

14.1 Introduction to Event and Timeout Concepts

This chapter provides an example of how to program a BPEL process service component to wait one minute for a response from a web service named Star Loan that provides loan offers. If Star Loan does not respond in one minute, then the BPEL process service component automatically selects an offer from another web service named United Loan. In the real world, the time limit is more like 48 hours. However, for this example, you do not want to wait that long to see if your BPEL process service component is working properly.

Because asynchronous web services can take a long time to return a response, a BPEL process service component must be able to time out, or give up waiting, and continue with the rest of the flow after a certain amount of time.

You can use a pick activity to configure a BPEL flow to either wait a specified amount of time or to continue performing its duties. To set an expiration period for the time, you can use the wait activity.

14.2 Creating a Pick Activity to Select Between Continuing a Process or Waiting

The pick activity provides two branches, each one with a condition. The branch that has its condition satisfied first is executed. In the following example, one branch's condition is to receive a loan offer, and the other branch's condition is to wait a specified amount of time.

Figure 14–1 provides an overview. The following activities take place (in order of priority):

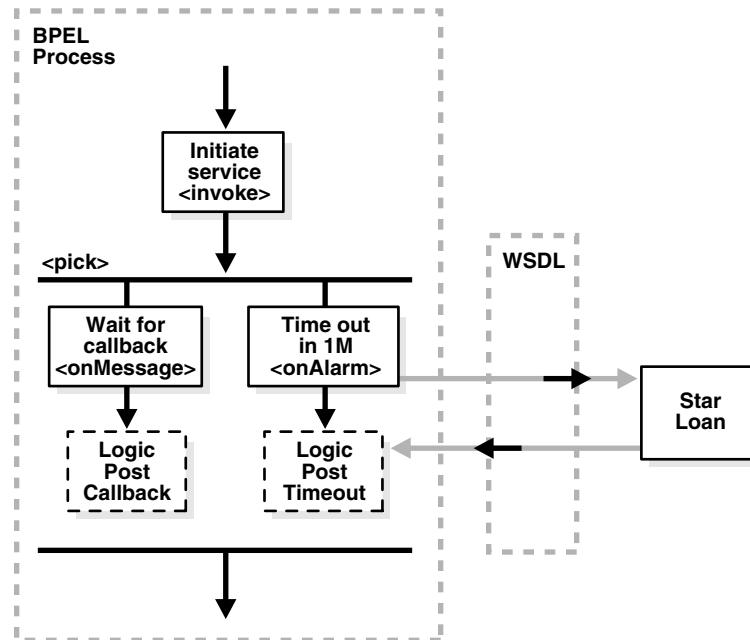
1. An invoke activity initiates a service, in this case, a request for a loan offer from Star Loan.
2. The pick activity begins next. It has the following conditions:
 - **onMessage**
This condition has code for receiving a reply in the form of a loan offer from the Star Loan web service. The onMessage code matches the code for receiving a response from the Star Loan web service before a timeout was added.
 - **onAlarm**
This condition has code for a timeout of one minute. This time is defined as PT1M, which means to wait one minute before timing out. In this timeout setting:
 - S stands for seconds
 - M for one minute
 - H for hour
 - D for day
 - Y for year

In the unlikely event that you want a time limit of 1 year, 3 days, and 15 seconds, you enter it as PT1Y3D15S. The remainder of the code sets the loan variables selected and approved to `false`, sets the annual percentage rate (APR) at 0.0, and copies this information into the `loanOffer` variable.

The time duration format is specified by the BPEL standard. For more detailed information on the time duration format, see the duration section of the most current *XML Schema Part 2: Datatypes* document at:

<http://www.w3.org/TR/xmlschema-2/#duration>

3. The pick activity condition that completes first is the one that the BPEL process service component executes. The other branch then is not executed.

Figure 14–1 Overview of the Pick Activity

An onMessage branch is similar to a receive activity in that it receives operations. However, you can define a pick activity with multiple onMessage branches that can wait for similar partner links and port types, but have different operations. Therefore, separate threads and parallel processes can be invoked for each operation. This differs from the receive activity in which there is only one operation. Another difference is that you can create a new instance of a business process with a receive activity (by selecting the **Create Instance** checkbox), but you cannot do this with a pick activity.

Note: You can also create onMessage branches in BPEL 1.1 scope activities and onAlarm branches in BPEL 1.1 and 2.0 scope activities. Expand the **Scope** activity in Oracle JDeveloper, and browse the icons on the left side to find the branch you want to add.

14.2.1 How To Create a Pick Activity

To create a pick activity:

1. In the SOA Composite Editor, double-click the BPEL process service component.
2. In the Component Palette, expand **BPEL Constructs**.
3. Drag a **Pick** activity into the designer.

The **Pick** activity includes an **onMessage** branch. [Figure 14–2](#) provides an example.

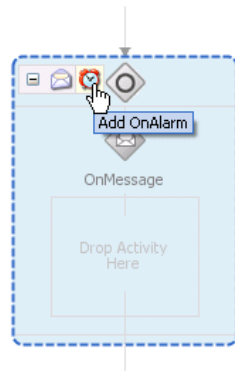
Figure 14–2 Pick Activity



4. Double-click the **onMessage** branch. [Figure 14–3](#) provides an example.

Figure 14–3 onMessage Branch

5. Edit its attributes to receive the response from the loan service.
6. Select the **Pick** activity.
Icons for adding additional **onMessage** branches and an **OnAlarm** branch are displayed.
7. Click **Add onAlarm**, as shown in [Figure 14–4](#).

Figure 14-4 onAlarm Branch Creation

An **OnAlarm** branch is displayed.

8. Double-click the **OnAlarm** branch of the **pick** activity and set its time limit to 1 minute. [Figure 14-5](#) provides an example.

Figure 14-5 OnAlarm Branch

9. Click **OK**.

14.2.2 What Happens When You Create a Pick Activity

The code segment in [Example 14-1](#) defines the `pick` activity for this operation after design completion:

Example 14-1 Pick Activity

```
<pick>
  <!-- receive the result of the remote process -->
  <onMessage partnerLink="LoanService"
    portType="services:LoanServiceCallback"
    operation="onResult" variable="loanOffer">

    <assign>
```

```

        <copy>
          <from variable="loanOffer" part="payload"/>
          <to variable="output" part="payload"/>
        </copy>
      </assign>

    </onMessage>
    <!-- wait for one minute, then timeout -->
    <onAlarm for="PT1M">
      <assign>
        <copy>
          <from>
            <loanOffer xmlns="http://www.autoloan.com/ns/autoloan">
              <providerName>Expired</providerName>
              <selected type="boolean">false</selected>
              <approved type="boolean">false</approved>
              <APR type="double">0.0</APR>
            </loanOffer>
          </from>
          <to variable="loanOffer" part="payload"/>
        </copy>
      </assign>
    </onAlarm>
  </pick>

```

14.2.3 What You May Need to Know About Simultaneous onMessage Branches in BPEL 2.0

Oracle BPEL Process Manager's implementation of BPEL 2.0 does not support simultaneous onMessage branches of a pick activity.

When a process has a pick activity with two onMessage branches as its starting activity (both with `initiate` set to `join` in their correlation definitions) and an invoking process that posts the invocations one after the other, it is assumed that both invocations reach the same instance of the invoked process. However, in Oracle BPEL Process Manager's implementation of BPEL 2.0, two instances of the invoked process are created for each invocation.

This is the expected behavior, but it differs from what is described in the BPEL 2.0 specification.

For example, assume you have synchronous BPEL process A, which has a flow activity with two parallel branches:

- Branch one invokes operation `processMessage1` on asynchronous BPEL process B.
- Branch two invokes operation `processMessage2` on asynchronous BPEL process B. The invocation occurs after a five second wait. BPEL process A then waits on a callback from BPEL process B and returns the output back to the client.

The idea is to create one instance of the invoked process and ensure that the second invocation happens after the first instance is already active and running.

BPEL process B has a pick activity with `createInstance` set to `yes`. The pick activity has two onMessage branches within it:

- One branch is for the `processMessage1` operation. For this operation, it goes to sleep for about 10 seconds.
- The other branch is for the `processMessage2` operation. For this operation, it waits for five seconds.

Both operations have the same input message type and correlation is defined with `initiate` set to `join`.

The expectation is that the `processMessage1` invocation is invoked immediately and the BPEL process B instance is created, which should sleep for ten seconds. After five seconds, the invoking process should then post the `processMessage2` invocation to BPEL process B and this invocation should go to the already existing instance instead of creating a new one (since the correlation ID is the same and `initiate` is set to `join`).

However, for each invocation, a new instance of BPEL process B is created and the result cannot be predicted.

- If the `processMessage2` operation branch finishes first, then the subsequent `assign` operation fails because the input variable from `processMessage1` is assumed to be null (for that instance).
- If the `processMessage1` operation branch finishes first, then the process returns callback data with only partial information (does not include the input from `processMessage2`).

In Oracle BPEL Process Manager's implementation, either one of the two operations (`processMessage1` or `processMessage2`) creates a new instance. This is implemented so that database queries do not need to be made to see if there are already instances created.

The workaround is to create two processes that are initiated by the two different operations.

14.3 Setting Timeouts for Request-Response Operations in Receive Activities

You can provide a timeout setting for request-response operations in receive activities. This provides an alternative to using the `onMessage` and `onAlarm` branches of a `pick` activity to specify a timeout duration for partner callbacks.

The following sections provide an overview of this functionality:

- Timeout settings relative from activity invocation
- Timeout settings as an absolute date time
- Timeout settings computed dynamically with an XPath expression
- `bpelx:timeout` fault thrown during an activity timeout
- Events added to the BPEL instance audit trail during an activity timeout
- Recoverable timeout activities during a server restart

Note: The timeout setting for request-response operations is not available in BPEL projects that support BPEL version 2.0.

14.3.1 Timeout Settings Relative from When the Activity is Invoked

You can specify a timeout setting relative from when the activity is invoked. This setting is specified as a relative duration using the syntax shown in [Example 14-2](#).

Example 14-2 Timeout Settings Relative from When the Activity is Invoked

```
<receive | bpelx:for="duration-expr">
```

```
        standard-elements  
    </receive>
```

This type uses the `bpelx:for` attribute to specify a static value or an XPath expression that must evaluate to an XML schema type duration. Only one of the `bpelx:for` or `bpelx:until` attributes is permitted for an activity.

If the XPath expression evaluates to a negative duration, the timeout is ignored and an event is logged to the instance audit trail indicating that the duration value is invalid.

Once a valid duration value is retrieved, the expiration date for the activity is set to the current node time (or cluster time after this is available), plus the duration value. For example, the duration value `bpelx:for=" 'PT5M' "` specifies that the activity expects an inbound message to arrive no later than five minutes after the activity has started execution.

Note: The timeout setting attribute does not apply to the `onMessage` branch of a pick activity because the same functionality currently exists with the `onMessage` and `onAlarm` branches of that activity.

Timeout durations can only be specified on the following:

- Midprocess receive activities
- Receive activities that do not specify `createInstance="true"`

A receive activity can only time out after it has been instantiated, which is not the case with entry receive activities.

14.3.2 Timeout Settings as an Absolute Date Time

You can specify a timeout setting as an absolute deadline for request-response receive activities. This type uses the syntax shown in [Example 14–3](#).

Example 14–3 *Timeout Settings as an Absolute Date Time*

```
<receive bpelx:until="deadline-expr">  
    standard-elements  
</receive>
```

The expected expiration time for the `bpelx:until` attribute must be at least two seconds ahead of the current time. Otherwise, the timer scheduling is ignored and skipped, just as if the timer was never specified.

The `bpelx:until` attribute specifies a static value or an XPath expression that must evaluate to an XML schema type `dateTime` or `date`. Only one of the `bpelx:for` or `bpelx:until` attributes is permitted for an activity.

XPath version 1.0 is not XML schema-aware. Therefore, none of the built-in functions of XPath version 1.0 can create or manipulate `dateTime` or `date` values. However, it is possible to perform one of the following:

- Write a constant (literal) that conforms to XML schema definitions and use that as a deadline value
- Extract a field from a variable (part) of one of these types and use that as a deadline value

XPath version 1.0 treats that literal as a string literal, but the result can be interpreted as a lexical representation of a `dateTime` or `date` value.

Once a valid `datetime` or `date` value has been retrieved, the expiration date for the activity is set to the specified date. For example, the `datetime` value `bpelx:until=" '2009-12-24T18:00+01:00' "` specifies that the activity expects an inbound message to arrive no later than Dec 24, 2009 6:00 pm UTC+1 after the activity has started execution.

Note: The timeout setting attribute does not apply to the `onMessage` branch of a pick activity because the same functionality currently exists with the `onMessage` and `onAlarm` branches of the pick activity.

Timeout dates can only be specified on the following activities:

- Midprocess receives
- Receive activities that do not specify `createInstance="true"`

A receive activity can only time out after it has been instantiated, which is not the case with entry receive activities.

14.3.3 Timeout Settings Computed Dynamically with an XPath Expression

The timeout setting for request-response receive and `onMessage` branches of pick activities can be set using an XPath expression instead of entering a static duration or `datetime` value. In this case, the value of the expression must return either:

- A string that can be interpreted as a static XML duration or `datetime` value
- An XML schema duration or `datetime` type

[Example 14-4](#) shows the syntax for using XPath expressions.

Example 14-4 Timeout Settings Computed Dynamically with an XPath Expression

```
<bpelx:for="bpws:getVariableData('input', 'payload',
'/tns:waitValue/tns:for')"/>

<bpelx:until="bpws:getVariableData('input', 'payload',
'/tns:waitValue/tns:until')"/>
```

If the returned expression value cannot be interpreted as an XML schema duration or `datetime` type, an event is logged in the instance audit trail indicating that an invalid duration and `datetime` value was specified, and no activity expiration time can be set.

14.3.4 bpelx:timeout Fault Thrown During an Activity Timeout

If a valid XML schema duration or `datetime` value is returned from the `bpelx:for` or `bpelx:until` attribute, a `bpelx:timeout` fault is thrown from the timed-out activity. This fault can be caught by any catch or catchAll block and handled like a regular BPEL fault. The message of the fault is the name of the activity. In addition, an event is logged to the instance audit trail indicating that the activity has timed out because the expected callback message failed to be received before the timeout duration.

If the activity receives a callback from the partner before the timeout period, no fault is thrown. If a callback is received while the activity is being timed out, the callback message is not delivered to the activity and it is marked as canceled in the delivery message table. If a timeout action is attempted at the same time that a callback

message is handled, the timeout action is ignored. As of 11g Release 1, instances are locked optimistically (as opposed to pessimistic locking in Release 10g). Therefore, the second action in line is still performed. However, the instance version check fails upon dehydration of the instance.

The `bpelx:timeout` fault can be thrown from a BPEL component if the component WSDL declares the fault on the operation. If the fault is not declared on the operation, the fault is converted into a `FabricInvocationException`, which is a runtime fault. This fault can be caught by any caller components (including BPEL components), but the fault type is no longer `bpelx:timeout` (however, the fault message string still indicates that the fault was originally a timeout fault).

14.3.5 Event Added to the BPEL Instance Audit Trail During an Activity Timeout

Once a `bpelx:timeout` fault is thrown from a timed-out activity, an event is logged to the instance audit trail indicating that the activity has timed out, as opposed to having received the expected callback message from its partner.

14.3.6 Recoverable Timeout Activities During a Server Restart (Refresh Expiration Alarm Table)

Activities that specify a valid timeout duration or `datetime` are likely implemented in a similar manner to `wait` and `onAlarm` activities with an expiration date for the underlying work item object. If the node that scheduled these activities with the scheduler goes down (either through graceful shutdown or abrupt termination), all these activities must be rescheduled with the scheduler upon server restart.

It is not possible to have a single node (the master node) in the cluster be responsible for rescheduling these activities upon node shutdown.

14.3.7 How to Set Timeouts for Request-Response Operations in Receive Activities

To set timeouts for request-response operations in receive activities:

1. In the SOA Composite Editor, double-click the version 1.1 BPEL process service component.
2. In the Component Palette, expand **BPEL Constructs**.
3. Drag a **Receive** activity into the designer.
4. Expand the activity.
5. Click the **Timeout** tab.

This tab enables you to set a timeout for request-response operations, as shown in [Figure 14–6](#).

Figure 14–6 Timeout Tab

The screenshot shows the 'Receive' dialog box with the 'Timeout' tab selected. The 'For' radio button is chosen, and the 'Time' section is set to 0 for all units (Yrs, Mons, Days, Hrs, Mins, Secs). The 'Expression' radio button is not selected. The dialog includes tabs for various configuration areas and standard action buttons at the bottom.

6. Specify appropriate values, and click **Apply**. For example:
 - To specify a timeout setting relative from when the activity is invoked, click the **For** button and enter a value or click the **Expression** button and specify an XPath expression.
 - To specify a timeout setting as an absolute deadline for a request-response operation, click the **Until** button and enter a value or click the **Expression** button and specify an XPath expression.
7. Click **Apply**, then **OK**.

14.3.8 What Happens When You Set Timeouts for Request-Response Operations in Receive Activities

The code segment in the `.bpel` file defines the specific operation after design completion.

For example, if you specified that the activity expects an inbound message to arrive no later than five minutes after the activity has started execution, the syntax displays as shown in [Example 14–5](#).

Example 14–5 Static Duration

```
bpelx:for=" 'PT5M' " />
```

For example, if you specified that the activity expects an inbound message to arrive no later than January 24, 2010 11:00 AM UTC+1 after the activity has started execution, the syntax displays as shown in [Example 14–6](#).

Example 14–6 datetime Value

```
bpelx:until=" '2010-01-24T11:00:00-08:00' " />
```

For example, if you specified an XPath expression to obtain a value for a timeout relative from when the activity is invoked, syntax similar to that shown in [Example 14-7](#) can display.

Example 14-7 XPath Expression

```
bpelx:for="bpws:getVariableData('inputVariable','payload','/tns:waitValue/tns:for')"/>
```

14.4 Creating a Wait Activity to Set an Expiration Time

The wait activity allows a process to wait for a given time period or until a time limit has been reached. Exactly one of the expiration criteria must be specified. A typical use of this activity is to invoke an operation at a certain time. You typically enter an expression that is dependent on the state of a process.

When specifying a time period for waiting, note the following:

- Wait times cannot be guaranteed if they are scheduled with other events that require processing. Due to this additional processing, the actual wait time can be greater than the wait time specified in the BPEL process.
- Wait times of less than two seconds are ignored by the server. Wait times above two seconds, but less than one minute, may not get executed in the exact, specified time. However, wait times in minutes do execute in the specified time.
- The default value of 2 seconds for wait times is specified with the **MinBPELWait** property in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control. You can set this property to any value and the wait delay is bypassed for any waits less than **MinBPELWait**.

Note: Quartz version 1.6 is supported for scheduling expiration events on wait activities.

14.4.1 How To Specify the Minimum Wait Time

You can specify the minimum time duration for a BPEL process to perform a wait that involves a dehydration. If the wait duration is less than or equal to the value, BPEL continues executing activities in the same thread and transaction.

1. From the **SOA Infrastructure** menu, select **SOA Administration > BPEL Properties**.
2. At the bottom of the BPEL Service Engine Properties page, click **More BPEL Configuration Properties**.
3. Click **MinBPELWait**.
4. In the **Value** field, specify a value in seconds.
5. Click **Apply**.
6. Click **Return**.

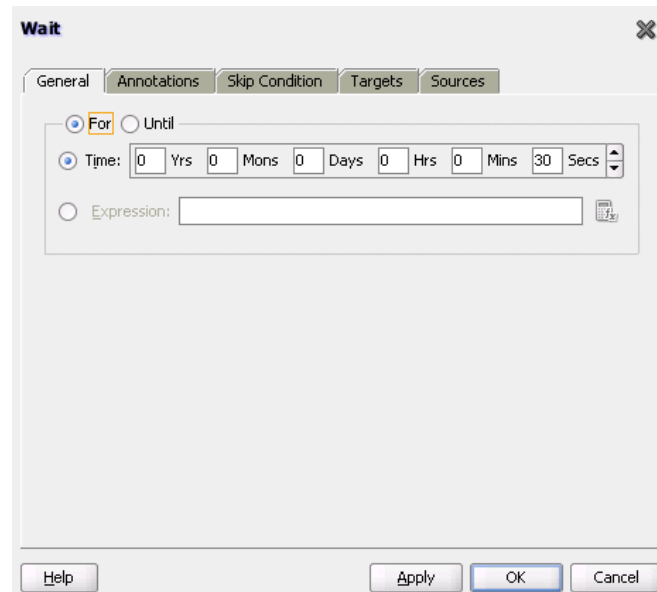
14.4.2 How to Create a Wait Activity

To create a wait activity:

1. In the Component Palette, expand **BPEL Constructs**.

2. Drag a **Wait** activity into the designer.
3. Double-click the **Wait** activity to display the Wait dialog.
4. In the **For** section, enter the amount of time for which to wait.
5. In the **Until** section, select the deadline for which to wait, as shown in [Figure 14-7](#).

Figure 14-7 Wait Dialog



14.4.3 What Happens When You Create a Wait Activity

Exactly one of the expiration criteria must be specified, as shown in [Example 14-8](#).

Example 14-8 Wait Activity

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
  standard-elements
</wait>
```

14.5 Specifying Events to Wait for Message Arrival with an OnEvent Branch in BPEL 2.0

You can create an onEvent branch in a scope activity that causes a specified event to wait for a message to arrive. For example, assume you have a credit request process that is initiated by a customer's credit request message. The request may be completely processed without the need for further interaction, and the results submitted to the customer. In some cases, however, the customer may want to inquire about the status of the credit request, modify the request content, or cancel the request entirely while it is being processed. You cannot expect these interactions to occur only at specific points in the business order processing. An event handler such as an onEvent branch enables the business process to accept requests (such as status request, modification request, or cancellation request) to arrive in parallel to the primary business logic flow.

The onEvent event handlers are associated with an enclosed scope. The onEvent event handlers are enabled when their scope is initialized and disabled when their scope

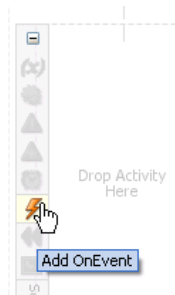
ends. When enabled, any number of events can occur. They are processed in parallel to the scope's primary activity and in parallel to each other. Message events also represent services operations exposed by a process and modeled as onEvent elements. Event handlers cannot create new process instances. Therefore, message events are always received by a process instance that is already active.

14.5.1 How to Create an onEvent Branch in a Scope Activity

To create an onEvent branch in a scope activity:

1. In the expanded **Scope** activity, click **Add OnEvent**, as shown in [Figure 14-8](#).

Figure 14-8 Add OnEvent Icon

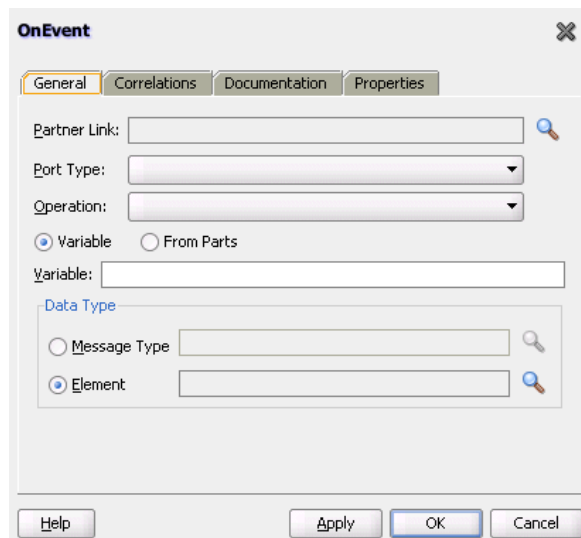


This creates an **OnEvent** branch and an enclosed scope activity.

2. Double-click the **OnEvent** branch.

The OnEvent dialog is displayed, as shown in [Figure 14-9](#).

Figure 14-9 OnEvent Dialog



3. In the **Partner Link** field, click the **Search** icon to select the partner link that contains the endpoint reference on which the message is expected to arrive.

The **Port Type** and **Operation** fields define the port type and operation invoked by the partner to cause the event.

4. Specify a method for receiving the message from the partner through use of a variable or **From Parts** elements.
5. Click **Apply**, then click **OK**.
6. Continue the design of your BPEL process.

14.5.2 What Happens When You Create an OnEvent Branch

[Example 14–9](#) provides an overview of `onEvent` branches in the `.bpel` file after design completion. The `onEvent` branches inquire about the status of the credit request, modify the request content, or cancel the request entirely while it is being processed

Example 14–9 *onEvent Branch*

```
<process name="creditRequestProcess" . . .>
  . . .
  <eventHandlers>
    <onEvent partnerLink="requestCreditScore"
      operation="queryCreditRequestStatus" . . .>
      <scope name="scopeStatus">...</scope>
    </onEvent>
    <onEvent partnerLink="requestCreditScore"
      operation="modifyCreditRequest" . . .>
      <scope name="scopeRequest">...</scope>
    </onEvent>
    <onEvent partnerLink="requestCreditScore"
      operation="cancelCreditRequest" . . .>
      <scope name="scopeCancel">...</scope>
    </onEvent>
  </eventHandlers>
  . . .
</process>
```

14.6 Setting Timeouts for Synchronous Processes

For synchronous processes that connect to a remote database, you must increase the **SyncMaxWaitTime** timeout property in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control.

For information on setting this property, see [Section 7.3, "Specifying Timeout Values."](#)

Coordinating Master and Detail Processes

This chapter describes how to coordinate master and detail processes in a BPEL process. This coordination enables you to specify the tasks performed by a master BPEL process and its related detail BPEL processes. This is sometimes referred to as a parent and child relationship.

This chapter includes the following sections:

- [Section 15.1, "Introduction to Master and Detail Process Coordinations"](#)
- [Section 15.2, "Defining Master and Detail Process Coordination in Oracle JDeveloper"](#)

15.1 Introduction to Master and Detail Process Coordinations

Master and detail coordinations consist of a one-to-many relationship between a single master process and multiple detail processes.

For example, assume a business process imports sales orders into an application. Each sales order consists of a header (customer information, ship-to address, and so on) and multiple lines (item name, item number, item quantity, price, and so on).

The following tasks are performed to execute the order:

- Validate the header. If the header is invalid, processing stops.
- Validate each line. If any lines are invalid, they are marked as invalid and processing stops.
- Perform inventory checks for each item. If an item is not available, a work order is created to assemble it.
- Stage items at the shipping dock after items for each line are available.
- Ship the order to the customer.

To perform these tasks, create a master process to check and validate each header and multiple BPEL processes to check and validate each line item.

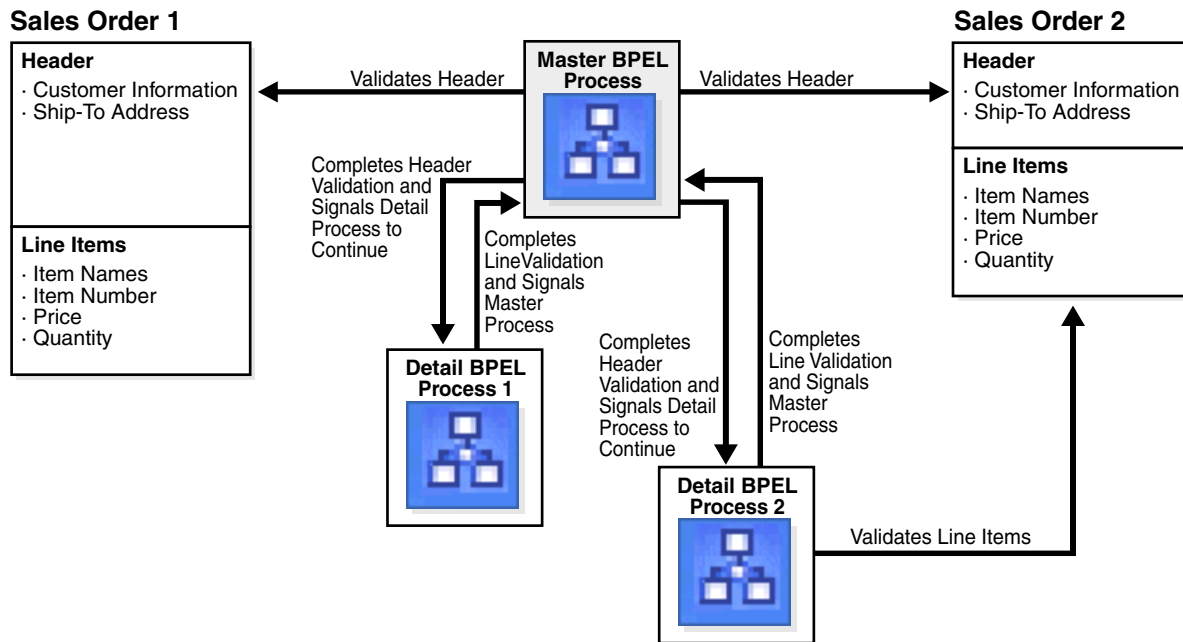
Potential coordination points are as follows:

- The master process must signal the detail processes that header validation is successful and to continue processing.
- Each detail process must signal the master process after line item validation is complete.
- Each detail process must signal the master process after the line item is available in inventory.

- After all line items are available, the master must signal each detail process to move its line item to the shipping dock (the dock may become too crowded if items are simply moved as soon as they are available).
- After all lines have been moved, the master process must execute logic to ship the fulfilled order to the customer.

Figure 15–1 provides an overview of the header and line item validation coordination points between one master process and two detail processes.

Figure 15–1 Master and Detail Coordination Overview (One BPEL Process to Two Detail Processes)

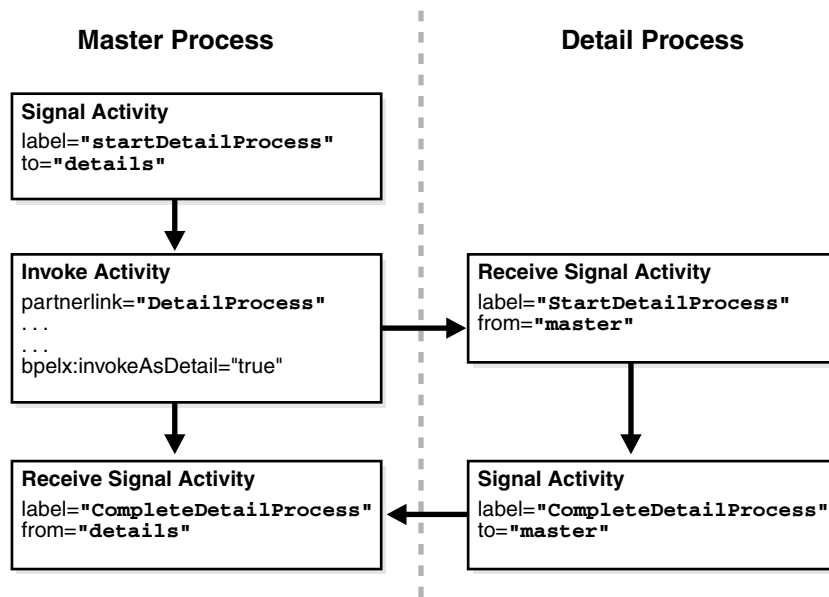


The following BPEL process activities coordinate actions between the master and detail processes:

- **signal:** notifies the other processes (master or detail) to continue processing
- **receive signal:** waits until it receives the proper notification signal from the other process (master or detail) before continuing its processing

Both activities are coordinated with label attributes defined in the BPEL process files. Labels are declared per master process definition.

Figure 15–2 provides an overview of the BPEL process flow coordination.

Figure 15–2 Master and Detail Syntax Overview (One BPEL Process to One Detail Process)

As shown in [Figure 15–2](#), each master and detail process includes a signal and receive signal activity. [Table 15–1](#) describes activity responsibilities based on the type of process in which they are defined.

Table 15–1 Master and Detail Process Coordination Responsibilities

If A...	Contains A...	Then...
Master process	Signal activity	The master process signals all of its associated detail processes at runtime.
Detail process	Receive signal activity	The detail process waits until it receives the signal executed by its master process.
Detail process	Signal activity	The detail process signals its associated master process at runtime that processing is complete.
Master process	Receive signal activity	The master process waits until it receives the signal executed by all of its detail processes.

If the signal activity executes before the receive signal activity, the state set by the signal activity is persisted and still effective for a later receive signal activity to read.

15.1.1 BPEL File Definition for the Master Process

The BPEL file for the master process defines coordination with the detail processes. The BPEL file shows that the master process interacts with the partner links of several detail processes. [Example 15–1](#) provides an example.

Example 15–1 BPEL File Definition for the Master Process

```

<process name="MasterProcess"
. . .
. . .
  <partnerLinks>
    <partnerLink name="client"
      partnerLinkType="tns:MasterProcess"
  
```

```

        myRole="MasterProcessProvider"
        partnerRole="MasterProcessRequester"/>
    <partnerLink name="DetailProcess"
        partnerLinkType="dp:DetailProcess"
        myRole="DetailProcessRequester"
        partnerRole="DetailProcessProvider"/>
    <partnerLink name="DetailProcess1"
        partnerLinkType="dp1:DetailProcess1"
        myRole="DetailProcess1Requester"
        partnerRole="DetailProcess1Provider"/>
    <partnerLink name="DetailProcess2"
        partnerLinkType="dp2:DetailProcess2"
        myRole="DetailProcess2Requester"
        partnerRole="DetailProcess2Provider"/>
</partnerLinks>

```

A signal activity shows the label value and the detail process coordinated with this master process. The label value (`startDetailProcess`) matches with the label value in the receive signal activity of all detail processes. This ensures that the signal is delivered to the correct process. There is one signal process per receive signal process. The master process signals all detail processes at runtime. This syntax shows a signal activity in a BPEL process that supports BPEL version 1.1.

```
<bpelx:signal name="notifyDetailProcess" label="startDetailProcess" to="details"/>
```

Note: In BPEL 2.0, the signal activity syntax is slightly different. The signal activity is wrapped in an `extensionActivity` element.

```

<extensionActivity>
    <bpelx:signal name="notifyDetailProcess"
        label="startDetailProcess" to="details"/>
</extensionActivity>

```

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.

In the invoke activity, ensure that the **Invoke As Detail** checkbox is selected. [Figure 15–3](#) provides details.

Figure 15–3 Invoke As Detail Checkbox

The image shows a user interface for configuring an invoke activity. It includes a text box for 'Conversation ID', another for 'Detail Label', and a checkbox labeled 'Invoke as Detail' which is currently checked. There is a small icon to the right of the text boxes.

This selection creates the partner process instance (`DetailProcess`) as a detail instance. You must select this checkbox in the invoke activity of the master process for each detail process with which to interact. [Example 15–2](#) provides an example of the BPEL file contents after you select the **Invoke As Detail** checkbox.

Example 15–2 *bpelx:invokeAsDetail Attribute*

```

<assign>
    <copy>
        <from variable="input" part="payload" query="/tns:processInfo/tns:value"/>
        <to variable="detail_input" part="payload" query="/dp:input/dp:number"/>
    </copy>
</assign>

```

```

    </copy>
  </assign>

  <invoke name="receiveInput" partnerLink="DetailProcess"
    portType="dp:DetailProcess"
    operation="initiate"
    inputVariable="detail_input"
    bpelx:invokeAsDetail="true"/>

  <!-- receive the result of the remote process -->
  <receive name="receive_DetailProcess" partnerLink="DetailProcess"
    portType="dp:DetailProcessCallback"
    operation="onResult" variable="detail_output"/>

```

The master BPEL process includes a receive signal activity. This activity indicates that the master process waits until it receives a signal from all of its detail processes. The label value (`detailProcessComplete`) matches with the label value in the signal activity of each detail process. This ensures that the signal is delivered to the correct process. [Example 15–3](#) provides an example. This syntax shows a receive signal activity in a BPEL process that supports BPEL version 1.1.

Example 15–3 Receive Signal Activity

```

<bpelx:receiveSignal name="waitForNotifyFromDetailProcess"
  label="detailProcessComplete"
  from="details"/>

```

Note: In BPEL 2.0, the receive signal activity syntax is slightly different. The receive signal activity is wrapped in an `extensionActivity` element.

```

<extensionActivity>
  <bpelx:receiveSignal name="waitForNotifyFromDetailProcess"
    label="detailProcessComplete" from="details"/>
</extensionActivity>

```

15.1.1.1 Correlating a Master Process with Multiple Detail Processes

For environments in which you have one master and multiple detail processes, use the `bpelx:detailLabel` attribute for signal correlation. [Example 15–4](#) shows how to use this attribute.

The first invoke activity invokes the `DetailsProcess` detail process and associates it with a label of `detailProcessComplete0`.

Example 15–4 First Invoke Activity

```

<invoke name="invokeDetailProcess" partnerLink="DetailProcess"
  portType="dp:DetailProcess"
  operation="initiate"
  inputVariable="detail_input"
  bpelx:detailLabel="detailProcessComplete0"
  bpelx:invokeAsDetail="true"/>

```

The second invoke activity invokes the `DetailsProcess1` detail process and associates it with a label of `detailProcessComplete1`. [Example 15–5](#) provides an example.

Example 15–5 Second Invoke Activity

```
<invoke name="invokeDetailProcess1" partnerLink="DetailProcess1"
  portType="dp1:DetailProcess1"
  operation="initiate"
  inputVariable="detail_input1"
  bpelx:detailLabel="detailProcessComplete1-2"
  bpelx:invokeAsDetail="true"/>
```

The third invoke activity invokes the DetailsProcess2 detail process again through a different port and with a different input variable. It associates the DetailsProcess2 detail process with a label of detailProcessComplete1-2, as shown in [Example 15–6](#).

Example 15–6 Third Invoke Activity

```
<invoke name="invokeDetailProcess2" partnerLink="DetailProcess2"
  portType="dp2:DetailProcess2"
  operation="initiate"
  inputVariable="detail_input2"
  bpelx:detailLabel="detailProcessComplete1-2"
  bpelx:invokeAsDetail="true"/>
```

The receive signal activity of the master process shown in [Example 15–7](#) waits for a return signal from detail process DetailProcess0.

Example 15–7 Receive Signal Activity

```
<!-- This is a receiveSignal waiting for 1 child to signal back -->
<bpelx:receiveSignal name="waitForNotifyFromDetailProcess0"
  label="detailProcessComplete0" from="details"/>
```

The second receive signal activity of the master process shown in [Example 15–8](#) also waits for a return signal from DetailProcess1 and DetailProcess2.

Example 15–8 Second Receive Signal Activity

```
<!-- This is a receiveSignal waiting for 2 child (detail) processes to signal back
-->
<bpelx:receiveSignal name="waitForNotifyFromDetailProcess1-2"
  label="detailProcessComplete1-2" from="details"/>
```

Note: If there is only one receive signal activity in the BPEL process, do not specify the `bpelx:detailLabel` attribute in the invoke activity. In these situations, a default `bpelx:detailLabel` attribute is assumed and does not need to be specified.

15.1.2 BPEL File Definition for Detail Processes

The BPEL process file of each detail process defines coordination with the master process.

A receive signal activity indicates that the detail process shown in [Example 15–9](#) waits until it receives a signal executed by its master process. The label value (`startDetailProcess`) matches with the label value in the signal activity of the master process.

Example 15–9 startDetailProcess Label Value

```
<bpelx:receiveSignal name="waitForNotifyFromMasterProcess"
  label="startDetailProcess" from="master" />
```

A signal activity indicates that the detail process shown in [Example 15–10](#) signals its associated master process at runtime that processing is complete. The label value (detailProcessComplete) matches with the label value in the receive signal activity of each master process.

Example 15–10 Signal Activity

```
<bpelx:signal name="notifyMasterProcess" label="detailProcessComplete"
  to="master" />
```

15.2 Defining Master and Detail Process Coordination in Oracle JDeveloper

This section provides an overview of how to define master and detail process coordination in Oracle BPEL Designer. In this example, one master process and one detail process are defined.

Note: This section only describes the tasks specific to master and detail process coordination. It does *not* describe the standard activities that you define in a BPEL process, such as creating variables, creating assign activities, and so on.

15.2.1 How to Create a Master Process

To create a master process:

1. In the SOA Composite Editor, create a BPEL process service component. For this example, the process is named **MasterProcess**.
2. Double-click the **MasterProcess** BPEL process.
3. In the Component Palette, expand **Oracle Extensions**.
4. Drag a **Signal** activity into the designer.
5. Double-click the **Signal** activity.

This activity signals the detail process to perform processing at runtime.

6. Enter the details described in [Table 15–2](#):

Table 15–2 Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, 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 6 on page 15-9.
To	Select details as the type of process to receive this signal.

[Figure 15–4](#) shows the Signal dialog.

Figure 15–4 Signal Dialog

7. Click **OK**.
8. Drag a **Receive Signal** activity into the designer.
9. Double-click the **Receive Signal** activity.

This activity enables the master process to wait until it receives the signal executed by all of its detail processes.

10. Enter the details shown in [Table 15–3](#):

Table 15–3 Receive Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, <code>waitForDetailProcess</code>).
Label	Enter a label name (for this example, <code>completeDetailProcess</code>). This label must match the signal activity label you set in the detail process in Step 10 on page 15-10.
To	Select details as the type of process from which to receive the signal.

[Figure 15–5](#) shows the Receive Signal dialog.

Figure 15–5 Receive Signal Dialog

The screenshot shows the 'Receive Signal' dialog box. The 'General' tab is selected. The 'Name' field contains 'waitForDetailProcess', the 'Label' field contains 'completeDetailProcess', and the 'From' dropdown menu is set to 'details'. The 'OK' button is highlighted.

11. Click OK.

The master process has now been designed to:

- Signal the detail process to perform processing at runtime.
- Wait until it receives the signal executed by the detail process.

15.2.2 How to Create a Detail Process

To create a detail process:

1. In the SOA Composite Editor, create a second BPEL process service component. For this example, the process is named **DetailProcess**.
2. Double-click the **DetailProcess** BPEL process.
3. In the Component Palette, expand **Oracle Extensions**.
4. Drag a **Receive Signal** activity into your BPEL process service component.
5. Double-click the **Receive Signal** activity.

This activity enables the detail process to wait until it receives the signal executed by its master process.

6. Enter the details shown in [Table 15–4](#):

Table 15–4 Receive Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, <code>WaitForContactFromMasterProcess</code>).
Label	Enter a label name (for this example, <code>beginDetailProcess</code>). This label must match the signal activity label you set in the master process in Step 6 on page 15-7.
To	Select master as the type of process from which to receive the signal.

[Figure 15–6](#) shows the Receive Signal dialog.

Figure 15–6 Receive Signal Dialog

7. Click **OK**.
8. Drag a **Signal** activity into the designer.
9. Double-click the **Signal** activity.

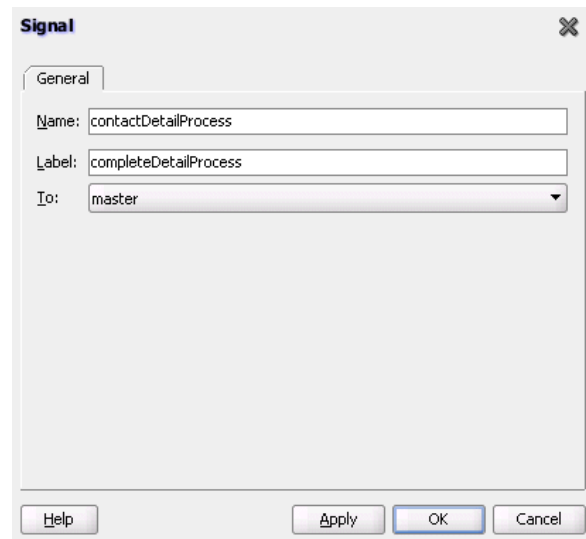
This activity enables the detail process to signal its associated master process at runtime that processing is complete.

10. Enter the details described in [Table 15–5](#):

Table 15–5 Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, <code>contactMasterProcess</code>).
Label	Enter a label name (for this example, <code>completeDetailProcess</code>). This label must match the receive signal activity label you set in the master process in Step 10 on page 15-8.
To	Select master as the destination.

[Figure 15–7](#) shows the Signal dialog.

Figure 15–7 Signal Dialog

11. Click **OK**.

The detail process has now been designed to:

- Wait until it receives the signal executed by its master process.
- Signal the master process at runtime that processing is complete.

15.2.3 How to Create an Invoke Activity

To create an invoke activity:

1. Return to the **MasterProcess** master process.
2. In the Component Palette, expand **BPEL Constructs**.
3. Drag an **Invoke** activity into your BPEL process service component.
4. Double-click the **Invoke** activity.
5. Select the **DetailProcess** BPEL process you created in Step 1 on page 15-9 as the partner link.
6. Complete all remaining fields in the Invoke dialog, and click **OK**.
7. In the designer, click **Source**.
8. Select the **Invoke As Detail** checkbox in the invoke activity. The BPEL file appears as shown in [Example 15–11](#).

Example 15–11 *bpelx:invokeAsdetail* Attribute

```
<invoke name="MyInvoke" partnerLink="DetailProcess"
  portType="dp:DetailProcess"
  operation="initiate"
  inputVariable="detail_input"
  bpelx:invokeAsDetail name="true"/>>
```

This attribute creates the partner process (DetailProcess) as a detail instance.

9. If this is an environment in which one master process is interacting with multiple detail processes, perform the following tasks:

- a. Specify the `bpelx:detailLabel` attribute for correlating with the receive signal activity, as shown in [Example 15–12](#).

Example 15–12 `bpelx:detailLabel` Attribute

```
<invoke name="MyInvoke" partnerLink="DetailProcess"
  portType="dp:DetailProcess"
  operation="initiate"
  inputVariable="detail_input"/>
bpelx:detailLabel="detailProcessComplete0"
<bpelx:invokeAsdetail name="true"/>
```

- b. Specify the same label value of `detailProcessComplete0` in the receive signal activity of the master process, as shown in [Example 15–13](#).

Example 15–13 `detailProcessComplete0` Label Value

```
<bpelx:receiveSignal name="waitForNotifyFromDetailProcess0-1"
label="detailProcessComplete0" from="details"/>
```

- c. Repeat these steps as necessary for additional detail processes, ensuring that you specify a different label value.
10. From the **File** main menu, select **Save All**.
Master and detail coordination design is now complete.

Using the Notification Service

This chapter describes how to send notifications from a BPEL process using a variety of channels. A BPEL process can be designed to send email, voice message, instant messaging (IM), or short message service (SMS) notifications. A BPEL process can also be designed to consider an end user's channel preference at runtime for selecting the notification channel.

This chapter includes the following sections:

- [Section 16.1, "Introduction to the Notification Service"](#)
- [Section 16.2, "Introduction to Notification Channel Setup"](#)
- [Section 16.3, "Selecting Notification Channels During BPEL Process Design"](#)
- [Section 16.4, "Allowing the End User to Select Notification Channels"](#)

Note: The fax and pager notification channels are not supported in 11g Release 1 (11.1.1).

16.1 Introduction to the Notification Service

Various scenarios may require sending email messages or other types of notifications to users as part of the process flow. For example, certain types of exceptions that cannot be handled automatically may require manual intervention. In this case, a BPEL process can use the notification service to alert users by voice, IM, SMS, or email.

The contact information (email address, phone number, and so on) of the recipient is either static (such as `admin@yourcompany.com`) or obtained dynamically during runtime. To obtain the contact information dynamically, XPath expressions can retrieve it from the identity store (LDAP) or extract it from the BPEL payload.

This chapter uses the following terms:

- Notification
An asynchronous message sent to a user by a specific channel. The message can be sent as an email, voice, IM, or SMS message.
- Actionable notification
A notification to which the user can respond. For example, workflow sends an email to a manager to approve or reject a purchase order. The manager approves or rejects the request by replying to the email with appropriate content.
- Human task email notification layer

Sends email notifications directly from a BPEL process or implicitly from the human task part of a BPEL process. Implicit notifications are modeled from the Human Task Editor.

For sending email notifications directly from a BPEL process, you must explicitly specify the user information in the BPEL process and can be inside or outside of a human task scope.

For sending email notifications implicitly from the human task part of a BPEL process, you only specify the recipient based on the relationship of the user with regards to the task (that is, the creator, assignee, and so on).

Note: Implicit notifications are processed through more layers of code than explicit notifications. If explicit notifications are functioning correctly, it does not mean that implicit notifications also function correctly.

- Oracle User Messaging Service

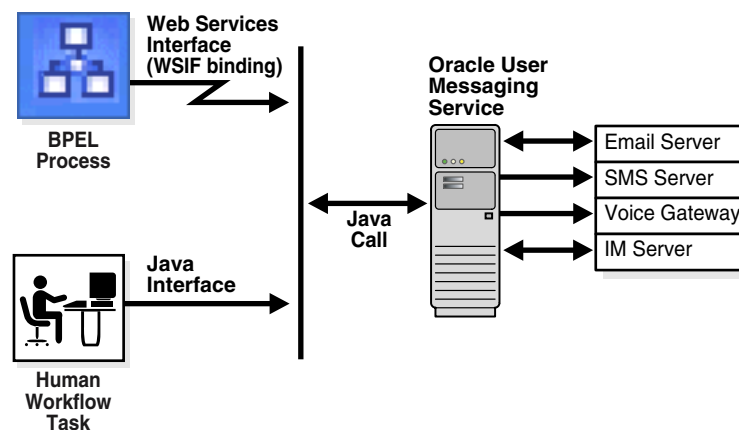
Oracle User Messaging Service is a new feature for release 11g. The BPEL notification service uses the underlying infrastructure provided by Oracle User Messaging Service to send notifications.

Oracle User Messaging Service also provides the user preference infrastructure for getting the end user's preferred channel during runtime.

For more information on the Oracle User Messaging Service, see [Appendix 59, "Oracle User Messaging Service."](#)

[Figure 16–1](#) shows the Oracle User Messaging Service interfaces and supported service types.

Figure 16–1 Service Interfaces and Supported Service Types



For more information about notifications, see the following section:

- [Section 31.2, "Notifications from Human Workflow"](#)
- [Section 27.3.10, "How to Specify Participant Notification Preferences"](#) for instructions on specifying email notifications in the Human Task Editor
- [Part XI, "Using Oracle User Messaging Service"](#)

16.2 Introduction to Notification Channel Setup

Notification setup is a multiple-step process that involves three user interface tools.

[Table 16–1](#) provides an overview of this process, including the task to perform, the tool to use, and the documentation to which to refer for more specific details.

Table 16–1 Notification Tasks

Task	Description	User Interface	Described In...
Select a channel for sending notifications in a SOA composite application.	Select a method for sending notifications: - Explicitly select and configure an email, IM, SMS, or voice channel. - or - Do not explicitly select a notification channel, but simply select that a notification must be sent. Channel selection occurs later in the User Messaging Preferences user interface.	Selected and configured by the BPEL process designer in Oracle BPEL Designer	Section 16.3, "Selecting Notification Channels During BPEL Process Design" or Section 16.4, "Allowing the End User to Select Notification Channels"
Configure the driver for the notification channel	You configure drivers on the same Oracle WebLogic Server on which you deploy the SOA composite application. This action enables participants to receive and forward notifications. Driver support is provided for email, IM, SMS, and voice channels.	Configured by the administrator in Oracle Enterprise Manager Fusion Middleware Control	<i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite</i>
Configure the notification mode and actionable accounts for human workflows	If you are using notifications with human workflow, you configure the notification mode and actionable account for email.	Configured by the administrator in Oracle Enterprise Manager Fusion Middleware Control	<i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite</i>
Register the devices used to access messages by specifying user preferences	This action enables workflow participants to receive notification messages. For example, the end user registers email clients and specifies the message content to receive and the channel to use for receiving messages. If no channel is specified, email is used by default. Note that the preferences set in this application are applicable only to that specific end user, and not to other users.	Registered by the end user in the User Messaging Preferences user interface. You can access this interface by selecting Preferences > Notification in Oracle BPM Worklist.	Part XI, "Using Oracle User Messaging Service"

16.3 Selecting Notification Channels During BPEL Process Design

Oracle JDeveloper includes the email, IM, SMS, and voice channel notification channels in the Component Palette. You can set the exact notification channels to use during design time. For example, a BPEL process can be designed to use the following notification channels:

- If an expense report amount is less than \$1000, an email notification channel is used.
- If an expense report amount is between \$1000 and \$2000, an SMS notification channel is used.

- If an expense report amount is more than \$2000, a voice notification channel is used.

To select the notification channel during BPEL process design:

1. In the Component Palette, expand **Oracle Extensions**.
2. Drag a notification channel into the designer:
 - **Email**
 - **IM**
 - **SMS**
 - **Voice**
3. See the section in [Table 16–2](#) based on the notification channel you selected.

Table 16–2 Notification Channels

If You Selected...	See...
Email	Section 16.3.1, "How To Configure the Email Notification Channel" to configure email notification
IM	Section 16.3.2, "How to Configure the IM Notification Channel" to configure IM notification
SMS	Section 16.3.3, "How to Configure the SMS Notification Channel" to configure SMS notification
Voice	Section 16.3.4, "How to Configure the Voice Notification Channel" to configure voice message notification

Note: If you delete an email, voice, SMS, or IM activity, any partner link with which it is integrated is not automatically deleted.

16.3.1 How To Configure the Email Notification Channel

When you select **Email** from the Component Palette, the Email dialog appears. [Figure 16–2](#) shows the required email notification parameters.

Figure 16–2 Email Dialog
To configure the email notification channel:

1. Enter information for each field as described in [Table 16–3](#).

Note: For the **To**, **CC**, and **Bcc** fields, separate multiple addresses with a semicolon (;).

Table 16–3 Email Notification Parameters

Name	Description
From Account	The name of the account used to send this message. The default account is named Default and is editable from the Workflow Notification Properties page in Oracle Enterprise Manager Fusion Middleware Control. To add additional accounts, you must use the System MBean Browser in Oracle Enterprise Manager Fusion Middleware Control. For information on editing this property in Oracle Enterprise Manager Fusion Middleware Control, see <i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite</i>.
To	The email address to which the message is to be delivered. This can be one of the following: ■ A static email address entered at the time the message is created ■ An email address retrieved using the identity service ■ A dynamic address from the payload The XPath Expression Builder can get the dynamic email address from the input. See Section 16.3.5, "How to Select Email Addresses and Telephone Numbers Dynamically."
CC and Bcc	The email addresses to which the message is copied and blind copied. This can be a static or dynamic address, as described for the To address.
Reply To	The email address to use for replies. This can be a static or dynamic address, as described for the To address.

Table 16–3 (Cont.) Email Notification Parameters

Name	Description
Subject	The subject of the email message. This can be plain text or dynamic text. The XPath Expression Builder can set dynamic text based on data from process variables that you specify.
Body	The message body of the email message. This can be plain text, HTML, or dynamic text, as described for the Subject parameter.
Multipart message with <i>n</i> attachments	Select to specify email attachments. See Section 16.3.1.1, "Setting Email Attachments." The number of attachments if Multipart message is selected. The number does not include the body. For example, if you have a body and one attachment, specify 1.

2. Click **OK**.

The BPEL fragment that invokes the notification service to send the email message is created.

3. See [Table 16–1](#) on page 16-3 for additional configuration procedures to perform.

The WebLogic Fusion Order Demo application uses an email activity in the **Scope_NotifyCustomerofCompletion** scope. The Oracle User Messaging Service sends the email to a customer when an order is fulfilled. The following details are specified in the Email dialog:

- An XPath expression specifies the customer's email address.

```
bpws:getVariableData('gCustomerInfoVariable','parameters','/ns3:findCustomerInfoV01CustomerInfoV0CriteriaResponse/ns3:result/ns2:ConfirmedEmail')
```

- A combination of manually-entered text and an XPath expression specifies the ID of the order:

```
Order with id
<%bpws:getVariableData('gOrderInfoVariable','/ns2:orderInfoV0SDO/ns2:OrderID')%> shipped!
```

- A combination of manually-entered text and an XPath expression specifies the body of the email message:

```
Dear
<%bpws:getVariableData('gCustomerInfoVariable','parameters','/ns6:findCustomerInfoV01CustomerInfoV0CriteriaResponse/ns6:result/ns4:FirstName')%>,
your order has been shipped.
```

[Figure 16–3](#) provides details.

Figure 16–3 Email Dialog

Email

General Attachments

From Account: Default

To: `teriarResponse/ns6:result/ns4:ConfirmedEmail')%>`

Cc:

Bcc:

Reply To:

Subject: `'iable','ns4:orderInfoVOSDO/ns4:OrderId')%> shipped!`

Body: `s4:FirstName')%>,
your order has been shipped.`

Message body can be plain text or HTML

Help Apply OK Cancel

16.3.1.1 Setting Email Attachments

You can send attachments with an email activity. Each attachment has three elements: name, MIME type, and value. All three elements must be set for each attachment.

To add an attachment to an email message:

1. From the Component Palette, select **Email** as the notification channel.
2. Specify values for **To**, **Subject**, and **Body**.
3. Click the **Attachments** tab.
4. Click the **Add** icon to add as many attachments as you require. (Note that the number of attachments does not need to include the body part.)
5. In the **Name** field, change the name or accept the default value of `Attachment $number$` .
6. In the **Mime Type** field, click the **Browse** icon to invoke the Expression Builder dialog for adding MIME type contents.
7. When complete, click **OK** to return to the **Attachments** tab.
8. In the **Value** field, click the **Browse** icon to invoke the Expression Builder dialog for adding the contents of the attachment.
9. When complete, click **OK** to return to the **Attachments** tab.

The BPEL fragment with an assign activity with multiple copy rules is generated. One of the copy rules copies the attachment.

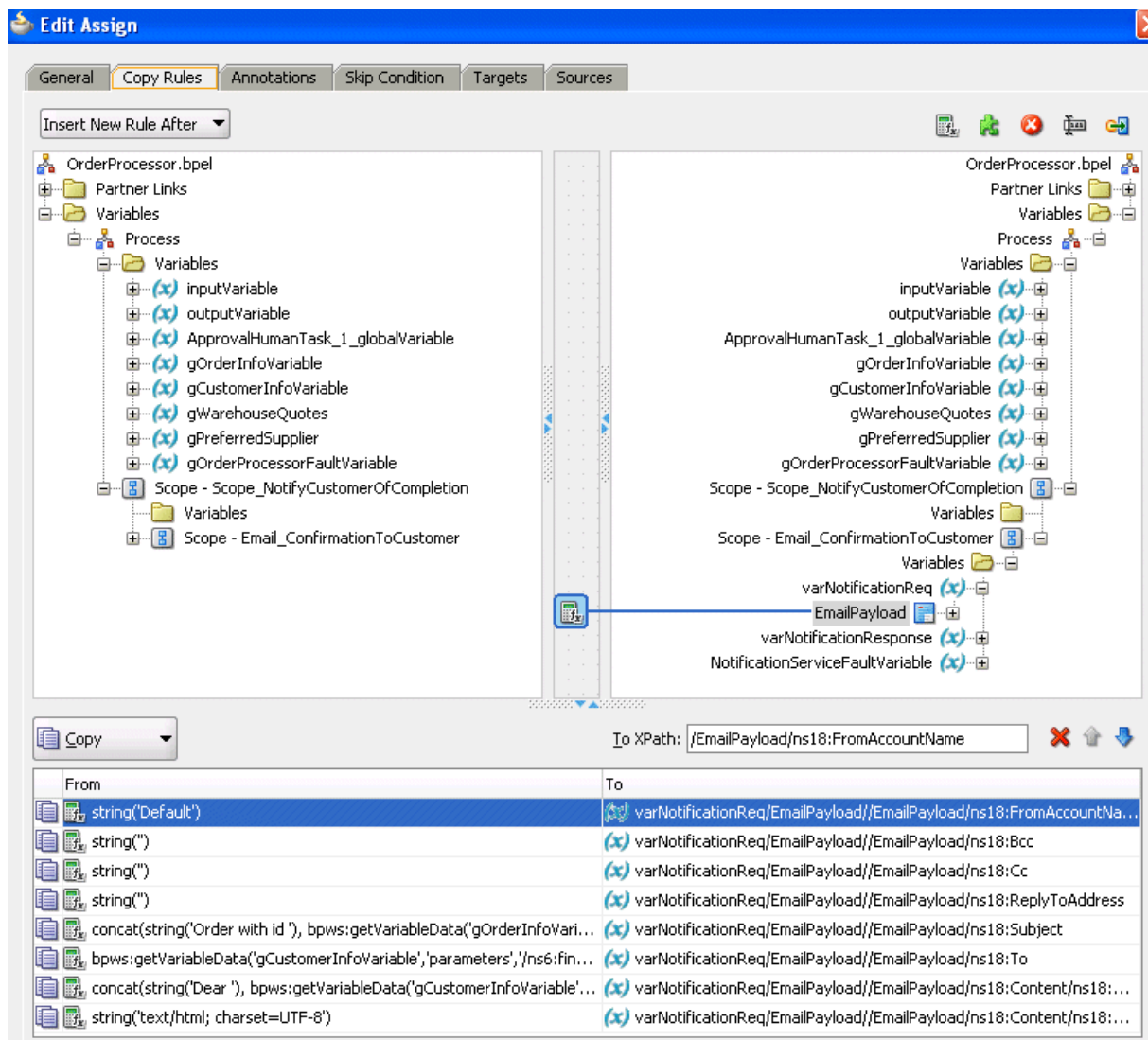
10. Click **OK**.

11. Expand the email activity.

Note that an assign activity named **EmailParamsAssign** appears.

12. Double-click **EmailParamsAssign**.

Note the settings in EmailParamsAssign, as shown in [Figure 16–4](#).

Figure 16–4 EmailParamsAssign Assign Activity

For more information about sending attachments using email, see the following documentation:

- [Appendix J, "Oracle User Messaging Service Applications"](#)
- The notification-101 sample, which is available at the following URL:

<https://soasamples.samplecode.oracle.com/>

16.3.1.2 Formatting the Body of an Email Message as HTML

You can format the body of an email message as HTML rather than as straight text. To perform this action, apply an XSLT transform to generate the email body. Add in the XSLT tag you want to use. Tools such as XMLSpy can provide assistance in writing and testing the XSLT. The MIME type should be `string('text/html; charset=UTF-8')`.

The email notification assignment looks as shown in [Example 16–1](#):

Example 16–1 Email Notification Assignment

```

<copy>
  <from
    expression="ora:processXSLT('TransformPositionSummary7.xslt',bpws:
    getVariableData('ClientPositionSummary'))"/>
    <to variable="varNotificationReq" part="EmailPayload"
    query="/EmailPayload/ns9:Content/ns9:ContentBody"/>
  </copy>

```

16.3.1.3 Using Dynamic HTML for Message Content Requires a CDATA Function

If the HTML for the message content of an email activity is generated dynamically, (as with XSLT, file read, and so on), it must be wrapped in a CDATA function. This prevents conflicts between the XML/HTML content of the message body and BPEL's internal XML data structures.

For example, assume you use the append operation shown in [Example 16–2](#) for the message content inside the email activity:

Example 16–2 Message Content Inside an Email Activity

```

<bpelx:append>
  <bpelx:from
    expression="ora:processXSLT('xsl/email.xslt',bpws:getVariableData('Variable_1'
    ))"/>
    <bpelx:to variable="varNotificationReq" part="EmailPayload"
    query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[1]
    /ns1:ContentBody"/>
  </bpelx:append>

```

For this to work correctly, you must pass the output of the `processXSLT()` function to the `CDATA()` function, as shown in [Example 16–3](#).

Example 16–3 CDATA() Function

```

<%ora:toCDATA(xdk:processXSLT('xsl/email.xslt',
  bpws:getVariableData('inputVariable','payload','/client:process/client:input')
))%>

```

16.3.2 How to Configure the IM Notification Channel

When you drag **IM** from the Component Palette, the IM dialog appears. [Figure 16–5](#) shows the required IM notification parameters.

Figure 16–5 IM Dialog
To configure the IM notification channel:

1. Enter information for each field as described in [Table 16–4](#).

Table 16–4 IM Notification Parameters

Name	Description
To	The IM address to which the message is to be delivered. Enter the address manually or click the XPath Expression Builder icon to display the Expression Builder dialog to dynamically enter an account.
Body	The IM message body. This can be plain text or dynamic text. The XPath Expression Builder can set dynamic text based on data from process variables that you specify.

2. Click **OK**.
The BPEL fragment that invokes the notification service for IM notification is created.
3. See [Table 16–1](#) on page 16-3 for additional configuration procedures to perform.

16.3.3 How to Configure the SMS Notification Channel

When you select **SMS** from the Component Palette, the SMS dialog appears. [Figure 16–6](#) shows the required SMS notification parameters.

Figure 16–6 SMS Dialog
To configure the SMS notification channel:

1. Enter information for each field as described in [Table 16–5](#).

Table 16–5 SMS Notification Parameters

Name	Description
From Number	The telephone number from which to send the SMS notification. This can be a static telephone number entered at the time the message is created or a dynamic telephone number from the payload. The XPath Expression Builder can get the dynamic telephone number from the input. See Section 16.3.5, "How to Select Email Addresses and Telephone Numbers Dynamically."
Telephone Number	Select a method for specifying the telephone number to which to deliver the message: ■ A static telephone number entered at the time the message is created. ■ A telephone number retrieved using the identity service. ■ A dynamic telephone number from the payload. The XPath Expression Builder can get the dynamic telephone number from the input.
Subject	The subject of the SMS message. This can be plain text or dynamic text. The XPath Expression Builder can set dynamic text based on data from process variables that you specify.
Body	The SMS message body. This must be plain text. This can be plain text or dynamic text as described for the Subject parameter.

2. Click **OK**.

The BPEL fragment that invokes the notification service for SMS notification is created.

3. See [Table 16–1](#) on page 16-3 for additional configuration procedures to perform.

16.3.4 How to Configure the Voice Notification Channel

When you select **Voice** from the Component Palette, the Voice dialog appears. [Figure 16–7](#) shows the required voice notification parameters.

Figure 16–7 Voice Dialog

To configure the voice notification channel:

1. Enter information for each field as described in [Table 16–6](#).

Table 16–6 Voice Notification Parameters

Name	Description
Telephone Number	The telephone number to which the message is to be delivered. Specify the number through one of the following methods: ■ A static telephone number entered at the time the message is created ■ A telephone number retrieved using the identity service ■ A dynamic telephone number from the payload The XPath Expression Builder can retrieve the dynamic telephone number from the input.
Body	The message body. This can be plain text, XML, or dynamic text. The XPath Expression Builder can set dynamic text based on data from process variables that you specify.

2. Click **OK**.
The BPEL fragment that invokes the notification service for voice notification is created.
3. See [Table 16–1](#) on page 16-3 for additional configuration procedures to perform.

16.3.5 How to Select Email Addresses and Telephone Numbers Dynamically

You may need to set email addresses or telephone numbers dynamically based on certain process variables. You can also look up contact information for a specific user using the built-in XPath functions for the identity service:

- To get the email address or telephone number directly from the payload, use the following XPath expression:

```
bpws:getVariableData('<variable name>', '<part>', 'input_xpath_to_get_an_address')
```

For example, to get the email address from variable `inputVariable` and part payload based on XPath `/client/BPELProcessRequest/client/mail`:

```
<%bpws:getVariableData('inputVariable', 'payload', '/client:BPELProcessRequest/client:email')%>
```

You can use the XPath Expression Builder to select the function and enter the XPath expression to get an address from the input variable.

- To get the email address or telephone number dynamically from the underlying identity store (LDAP) use the following XPath expression:

```
ids:getUserProperty(userName, attributeName[, realmName])
```

The first argument evaluates to the user ID. The second argument is the property name. The third argument is the realm name. [Table 16–7](#) lists the property names that can be used in this XPath function.

Table 16–7 Properties for the Dynamic User XPath Function

Property Name	Description
mail	Look up a user's email address.
telephoneNumber	Look up a user's telephone number.
mobile	Look up a user's mobile telephone number.
homephone	Look up a user's home telephone number.

The following example gets the email address of the user identified by the variable `inputVariable`, part payload, and queries

```
/client:BPELProcessRequest/client:userID:
```

```
ids:getUserProperty(bpws:getVariableData('inputVariable', 'payload', '/client:BPELProcessRequest/client:userid'), 'mail')
```

If `realmName` is not specified, then the default realm name is used. For example, if the default realm name is `jazn.com`, the following XPath expression searches for the user in the `jazn.com` realm:

```
ids:getUserProperty('jcooper', 'mail');
```

The following XPath expression provides the same functionality as the one above. In this case, however, the realm name of `jazn.com` is explicitly specified:

```
ids:getUserProperty('jcooper', 'mail', 'jazn.com');
```

16.3.6 How to Select Notification Recipients by Browsing the User Directory

You can select users or groups in Oracle JDeveloper to whom you want to send notifications by browsing the user directory (for example, Oracle Internet Directory) that is configured for use with Oracle BPEL Process Manager. Click the **Search** icon to the right of the following fields to open the Identity Lookup dialog:

- **To** field on the Email and IM dialogs

- **Telephone Number** field on the SMS and Voice dialogs

For more information about using the Identity Lookup dialog, see [Chapter 31, "Introduction to Human Workflow Services"](#)

16.4 Allowing the End User to Select Notification Channels

You can design a BPEL process in which you do not explicitly select a notification channel during design time, but simply indicate that a notification must be sent. The channel to use for sending notifications is resolved at runtime based on preferences defined by the end user in the User Messaging Preferences user interface of the Oracle User Messaging Service. This moves the responsibility of notification channel selection from the BPEL process designer in Oracle BPEL Designer to the end user. If the end user does not select a preferred channel or rule, email is used by default for sending notifications to that user. Regardless of who selects the channel to use, channel use is still based on the driver installation and configuration performed in the Oracle User Messaging Service section of Oracle Enterprise Manager Fusion Middleware Control by the administrator.

For example, an end user may set their preferences as follows:

- If an expense report amount is less than \$153, they receive an email notification.
- If an expense report amount is between \$153 and \$3678, they receive an SMS notification.
- If an expense report amount is more than \$3678, they receive a voice notification.

Note: You can also set user preferences for sending notifications in human workflows in the Human Task Editor. Set these preferences in the **Notification Filters** part of the **Notification Settings** section. These preferences are used to evaluate rules in the task. For more information, see [Section 27.3.10.8, "Sending Task Attachments with Email Notifications."](#)

For more information about the User Messaging Preferences user interface, see [Chapter 64, "User Messaging Preferences."](#)

16.4.1 How to Allow the End User to Select Notification Channels

To allow the end user to select notification channels:

1. From the Component Palette list, select **BPEL**.
2. Expand **BPEL Activities and Components**.
3. From the Component Palette, drag the **User Notification** activity into the designer. [Figure 16–8](#) shows the required user notification parameters.

Figure 16–8 User Notification Dialog

User Notification

General Advanced Skip Condition

To: jcooper  
Use commas to separate multiple names

Subject: Vacation Request 

Notification Message: 
 Please approve the vacation request

Applies to Voice, SMS, Email and IM.

Help Apply OK Cancel

4. Enter information for each field as described in [Table 16–8](#).

Table 16–8 User Notification Parameters

Name	Description
To	Enter a valid user for the recipient of this notification message through one of the following methods: ■ Enter the user manually ■ Click the Search icon to display a dialog for selecting a user configured through the identity service. The identity service enables the lookup of user properties, roles, and group memberships. ■ Click the XPath Expression Builder icon to display the Expression Builder dialog to dynamically enter a user. Note: You must specify a user name (for example, jcooper) instead of an address.
Subject	Enter a message name or click the XPath Expression Builder icon to display the Expression Builder dialog to dynamically enter a subject. If notification is sent through email, this field is used during runtime. This field is ignored if notifications are sent through the voice, SMS, or IM channels.
Notification Message	Enter the notification message or click the XPath Expression Builder icon to display the Expression Builder dialog to dynamically enter a message to send.

5. Click **Apply**.

16.4.1.1 How to Create and Send Headers for Notifications

The **Advanced** tab of the User Notification dialog enables you to create and send header and name information that may be useful to an end user in creating their own preference rules for receiving notifications. For example:

- The BPEL designer creates specifies the users named jcooper and jstein in the **General** tab.

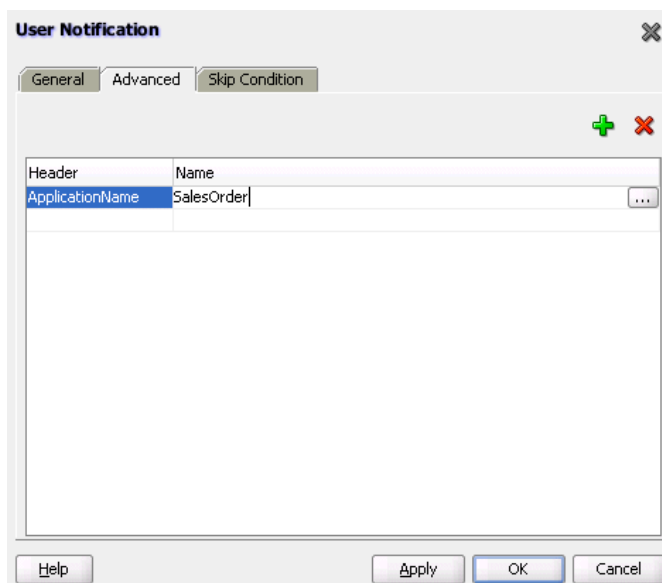
- The BPEL designer creates the following header and name information in the **Advanced** tab:
 - Amount = payload->salary
 - Application = HR-Application
 - The administrator deploys the process and configures various channel drivers in Oracle Enterprise Manager Fusion Middleware Control.
 - The end user jcooper creates the following preference rules in the User Messaging Preferences user interface:


```
'Email if Amount < 30000' and 'SMS if Amount is between 30000 and 100000' and
            'Voice if Amount > 100000'
```
 - The end user jstein creates the following preference rule in the User Messaging Preferences user interface:


```
If "Application == HR-Application" and Amount > 2000000" send Voice
```
1. If you want to create and send header and name information to an end user for creating their own preference rules, click **Advanced**.

Figure 16–9 shows the **Advanced** tab of the User Notification dialog.

Figure 16–9 User Notification Advanced Parameters



2. Click the **Add** icon to add a row to the **Header** and **Name** columns.
3. In the **Header** column, click the field to display a list for selecting a value. Otherwise, manually enter a value.
4. In the **Name** column, enter a value.
5. Click **OK**.

Using Oracle BPEL Process Manager Sensors

This chapter describes how to use sensors to select BPEL activities, variables, and faults to monitor during runtime. This chapter describes how to use and set up sensors for a BPEL process.

This chapter includes the following sections:

- [Section 17.1, "Introduction to Sensors"](#)
- [Section 17.2, "Configuring Sensors and Sensor Actions in Oracle JDeveloper"](#)
- [Section 17.3, "Viewing Sensors and Sensor Action Definitions in Oracle Enterprise Manager Fusion Middleware Control"](#)

For more information about sensors, see the following sections:

- [Section 50.7, "Integrating BPEL Sensors Using Oracle BAM Sensor Action"](#) for how to create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects in an Oracle BAM Server
- [Appendix D, "Understanding Sensor Public Views and the Sensor Actions XSD"](#)

17.1 Introduction to Sensors

Sensors are used to declare interest in specific events throughout the life cycle of a BPEL process instance. In a business process, that can be the activation and completion of a specific activity or the modification of a variable value in the business process.

When a sensor is triggered, a specific sensor value is created. For example, if a sensor declares interest in the completion of a BPEL scope, the sensor value consists of the name of the BPEL scope and a time stamp value of when the activity was completed. If a sensor value declares interest in a BPEL process variable, then the sensor value consists of the value of the variable at the moment it was modified, a time stamp when the variable was modified, and the activity name and type that modified the BPEL variable.

The data format for sensor values is normalized and well-defined using XML schema.

A sensor action is an instruction on how to process sensor values. When a sensor is triggered by Oracle BPEL Process Manager, a new sensor value for that sensor is created. After that, all the sensor actions associated with that sensor are performed. A sensor action typically persists the sensor value in a database or sends the normalized sensor value data to a JMS queue or topic. For integration with Oracle Business Activity Monitoring, the sensor value can be sent to the BAM adapter.

You can define the following types of sensors, either through Oracle JDeveloper or manually by providing sensor configuration files.

- Activity sensors

Activity sensors are used to monitor the execution of activities within a BPEL process. For example, they can 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 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.

These sensors are exposed through the following public SQL views:

- BPEL_ACTIVITY_SENSOR_VALUES
- BPEL_FAULT_SENSOR_VALUES
- BPEL_VARIABLE_SENSOR_VALUES

These views can be joined with the BPEL_PROCESS_INSTANCES view to associate the sensor value with the BPEL process instance that created the sensor values. For more information, see [Appendix D, "Understanding Sensor Public Views and the Sensor Actions XSD."](#)

When you model sensors in Oracle JDeveloper, two new files are created as part of the BPEL process archive:

- *bpel_process_name_sensor.xml*
Contains the sensor definitions of a BPEL process
- *bpel_process_name_sensorAction.xml*
Contains the sensor action definitions of a BPEL process

See [Section 17.2.2, "How to Configure Sensors"](#) and [Section 17.2.3, "How to Configure Sensor Actions"](#) for how these files are created.

After you define sensors for a BPEL process, you must configure sensor actions to publish the sensor data to a specified destination. If no sensor action is defined for a sensor, then nothing happens at runtime.

The following information is required for a sensor action:

- Name
- Publish type

The publish type specifies the destination in which the sensor data must be presented. You can publish sensor data to the following destination types.

- Database

Publishes the sensor data to the reports schema in the database. The sensor data can then be queried using SQL.

- JMS queue
Publishes the sensor data to a JMS queue. The XML data is posted in accordance with the `Sensor.xsd` file. This file is included with Oracle JDeveloper in the `JDEV_HOME\jdeveloper\integration\seed\soa\shared\bpel` directory.
- JMS topic
Publishes the sensor data to a JMS topic. The XML data is posted in accordance with the same `Sensor.xsd` file used with JMS queues.
- Custom
Publishes the data to a custom Java class.
- JMS Adapter
Uses the JMS adapter to publish to remote queues or topics and a variety of different JMS providers. The JMS queue and JMS topic publish types only publish to local JMS destinations.
- List of sensors
The sensors for a sensor action.

Oracle BAM sensors publish information and events from Oracle BPEL Process Manager to Oracle BAM. Oracle BAM can display the data in rich real-time dashboards for end-to-end monitoring of an application. For more information, see [Section 50.7, "Integrating BPEL Sensors Using Oracle BAM Sensor Action."](#)

17.2 Configuring Sensors and Sensor Actions in Oracle JDeveloper

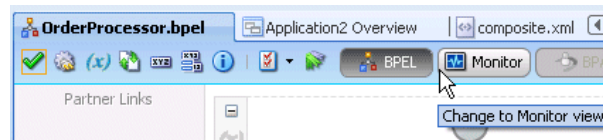
In Oracle JDeveloper, sensor actions and sensors are displayed as part of **Monitor** view.

17.2.1 How to Access Sensors and Sensor Actions

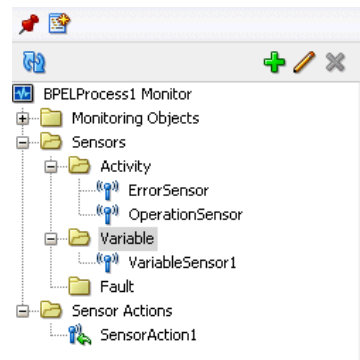
To access sensors and sensor actions:

1. Select **Monitor** at the top of Oracle BPEL Designer, as shown in [Figure 17-1](#).

Figure 17-1 Monitor View

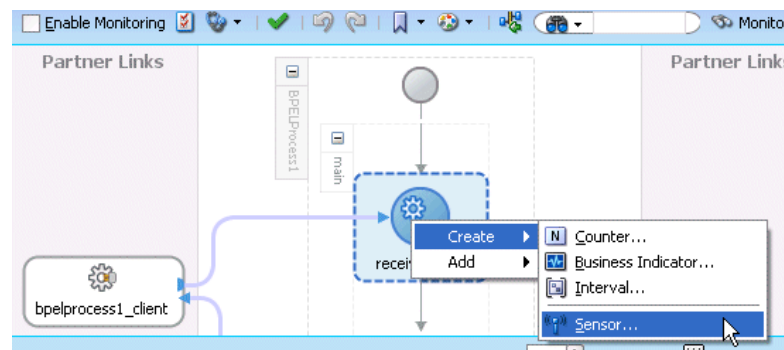


[Figure 17-2](#) shows the sensor actions and sensors in the Structure window.

Figure 17–2 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.

2. Add sensor actions by right-clicking the **Sensor Actions** folders and selecting **Create > Sensor Action**.
3. Add activity sensors, variable sensors, or fault sensors as follows:
 - a. Expand the **Sensors** folder.
 - b. Right-click the appropriate **Activity**, **Variable**, or **Fault** subfolder.
 - c. Click **Create**.
4. Add sensors to individual activities by right-clicking an activity and selecting **Create > Sensor**. Figure 17–3 provides details.

Figure 17–3 Creating an Activity Sensor

The following sections describe how to configure sensors and sensor actions.

17.2.2 How to Configure Sensors

This section describes how to configure activity, variable, and fault sensors.

To configure an activity sensor:

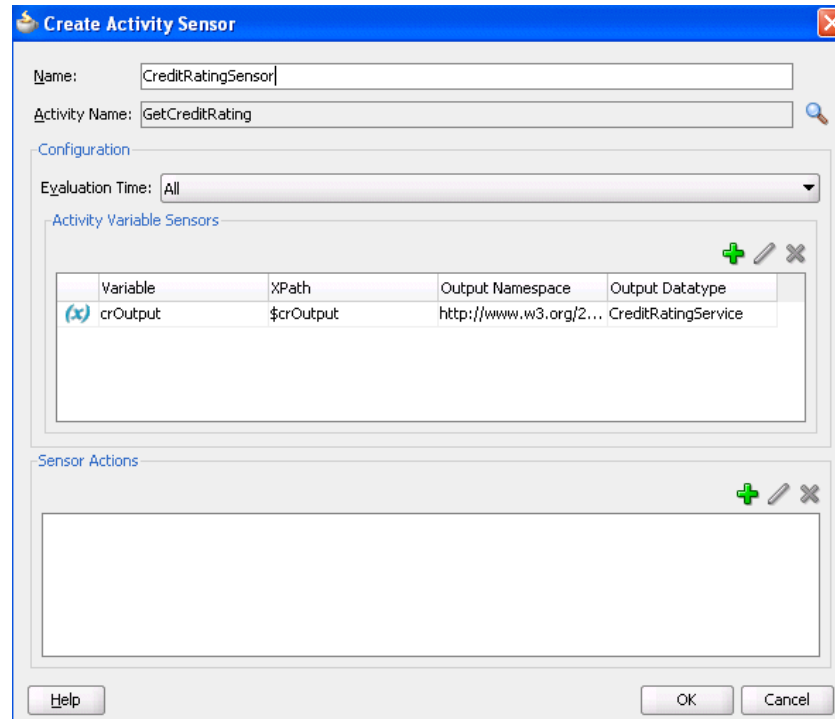
Assume you are monitoring a loan flow application, and want to know the following:

- When a scope named GetCreditRating is initiated
- When it is completed
- At completion, what is the credit rating for the customer

The solution is to create an activity sensor for the GetCreditRating scope in Oracle BPEL Designer, as shown in [Figure 17-4](#).

1. Select **Monitor** at the top of Oracle BPEL Designer.
2. In the Structure window, expand the **Sensors** folder.
3. Right-click **Activity**, and select **Create**.
4. To the right of the **Activity Name** field, click the **Browse** icon to select the activity for which to create the sensor. This is a required field.

Figure 17-4 *Creating an Activity Sensor*



Activities that have sensors associated with them are identified with a magnifying glass in Oracle BPEL Designer.

The **Evaluation Time** list shown in [Figure 17-4](#) controls the point at which the sensor is fired.

5. Select from the following:
 - **All:**
The sensor monitors during the activation, completion, fault, compensation, and retry phases.
 - **Activation**
The sensor is fired just before the activity is executed.
 - **Completion**
The sensor is fired just after the activity is executed.
 - **Fault**
The sensor is fired if a fault occurs during the execution of the activity. Select this value only for sensors that monitor simple activities.

- **Compensation**

The sensor is fired when the associated scope activity is compensated. Select this value only for sensors that monitor scopes.

- **Retry**

The sensor is fired when the associated invoke activity is retried.

A new entry is created in the *bpel_process_name_sensor.xml* file, as shown in [Example 17-1](#):

Example 17-1 *bpel_process_name_sensor.xml* file

```
<sensor sensorName="CreditRatingSensor"

classname="oracle.tip.pc.services.reports.dca.agents.BpelActivitySensorAgent"
    kind="activity"
    target="GetCreditRating">

<activityConfig evalTime="all">
    <variable outputNamespace="http://www.w3.org/2001/XMLSchema"
        outputDataType="int"
        target="$crOutput/payload//services:rating"/>
</activityConfig>
</sensor>
```

6. If you want to create a variable sensor on the activity, then in the **Activity Variable Sensors** section, click the **Add** icon. This is an optional field.

To configure a variable sensor:

If you want to record all the incoming loan requests, you can create a variable sensor.

1. In the Structure window, expand the **Sensors** folder.
2. Right-click **Variable**, and select **Create**.
3. Click the **Edit** icon to the right of the **Target** field to create a variable sensor for a variable (for this example, named **input**), as shown in [Figure 17-5](#).

Figure 17–5 Creating a Variable Sensor

Based on your selection for the **Target** field, the **Output Namespace** and **Output Datatype** fields are automatically filled in.

A new entry is created in the `bpel_process_name_sensor.xml` file, as shown in [Example 17–2](#):

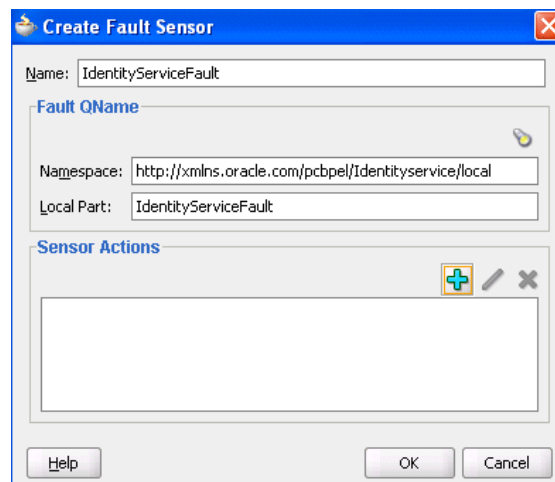
Example 17–2 `bpel_process_name_sensor.xml` file

```
<sensor sensorName="LoanApplicationSensor"
  classname="oracle.tip.pc.services.reports.dca.agents.BpelVariableSensorAgent"
  kind="variable"
  target="$input/payload">
  <variableConfig outputNamespace="http://www.autoloan.com/ns/autoloan"
    outputDataType="loanApplication"/>
</sensor>
```

To configure a fault sensor:

If you want to monitor faults (for this example, from the identity service), you can create a fault sensor.

1. In the Structure window, expand the **Sensors** folder.
2. Right-click **Fault**, and select **Create**.
3. Click the **Browse** icon above the **Namespace** field to select to create a fault sensor, as shown in [Figure 17–6](#).

Figure 17–6 Creating a Fault Sensor

Based on your selection, the **Namespace** and **Local Parts** fields are automatically filled in.

A new entry is created in the `bpel_process_name_sensor.xml` file, as shown in [Example 17–3](#):

Example 17–3 `bpel_process_name_sensor.xml` file

```
<sensor sensorName="IdentityServiceFault"
        classname="oracle.tip.pc.services.reports.dca.agents.BpelFaultSensorAgent"
        kind="fault"
        target="is:identityServiceFault">
  <faultConfig/>
</sensor>
```

17.2.3 How to Configure Sensor Actions

When you create sensors, you identify the activities, variables, and faults you want to monitor during runtime. If you want to publish the values of the sensors to an endpoint (for example, you want to publish the data of the **LoanApplicationSensor** variable sensor created in [Figure 17–5](#) to a JMS queue), then create a sensor action, as shown in [Figure 17–7](#), and associate it with the **LoanApplicationSensor** variable.

To configure a sensor action:

1. In the Structure window, right-click the **Sensor Actions** folder.
2. Select **Create > Sensor Action**.
3. Enter the details described in [Table 17–1](#).

Table 17–1 Sensor Actions Dialog

Field	Description
Name	Enter a name or accept the default name.
Publish Type	Select the destination to which to publish sensor data. For more information, see Section 17.1, "Introduction to Sensors."
JMS Connection Factory	If your publish type is JMS Queue , JMS Topic , or JMS Adapter , specify the connection factory.

Table 17–1 (Cont.) Sensor Actions Dialog

Field	Description
Publish Target	If your publish type is JMS Queue, JMS Topic, Custom, or JMS Adapter, specify the publish target. The publish target represents different things depending on the publish type specified: ■ If the publish type is a database, this field is left blank. ■ If the publish type is JMS Queue, JMS Topic, or JMS Adapter, this represents the JMS destination's JNDI name. ■ If the publish type is Custom, this represents the fully-qualified Java class name.
Filter	Enter filter logic as a boolean expression. A filter enables you to monitor sensor data within a specific range. For an example of a configured filter, see Figure 17–9 and Example 17–6 .
Enable	Deselect this checkbox to disable a sensor action. By default, sensor actions are enabled. If you disable a sensor action by deselecting this checkbox, the action does not publish data.

Figure 17–7 Creating a Sensor Action

A new entry is created in the `bpel_process_name_sensorAction.xml` file, as shown in [Example 17–4](#):

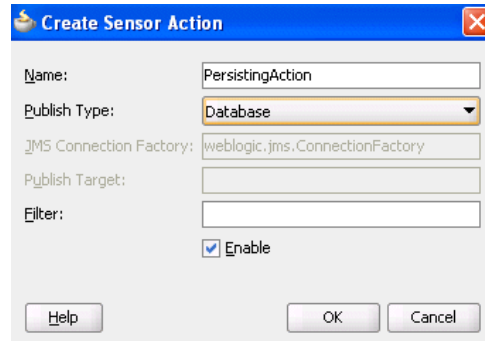
Example 17–4 `bpel_process_name_sensorAction.xml` file

```
<action name="BAMFeed"
  enabled="true"
  publishType="JMSQueue"
  publishTarget="jms/bamTopic">
  <sensorName>LoanApplicationSensor</sensorName>
  <property name="JMSConnectionFactory">
    weblogic.jms.ConnectionFactory
  </property>
</action>
```

Note: You cannot specify a < (less than) sign in the **Filter** field of the Sensor Action dialog. If you do, Oracle JDeveloper translates the < sign to `<` in the `bpel_process_name_sensorAction.xml` file. In addition, you cannot specify a < sign by directly editing the `filename_sensorAction.xml` file. This action causes an error.

If you want to publish the values of **LoanApplicationSensor** and **CreditRatingSensor** to the reports schema in the database, create an additional sensor action, as shown in [Figure 17-8](#), and associate it with both **CreditRatingSensor** and **LoanApplicationSensor**.

Figure 17-8 Creating an Additional Sensor Action



A new entry is created in the `bpel_process_name_sensorAction.xml` file, as shown in [Example 17-5](#):

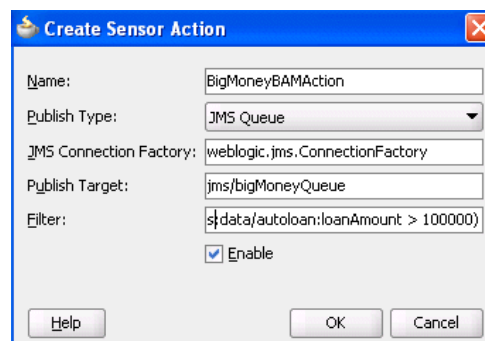
Example 17-5 `bpel_process_name_sensorAction.xml` file

```
<action name="PersistingAction"
  enabled="true"
  publishType="BPELReportsSchema">
  <sensorName>LoanApplicationSensor</sensorName>
  <sensorName>CreditRatingSensor</sensorName>
</action>
```

The data of one sensor can be published to multiple endpoints. In the two preceding code samples, the data of **LoanApplicationSensor** was published to a JMS queue and to the reports schema in the database.

If you want to monitor loan requests for which the loan amount is greater than \$100,000, you can create a sensor action with a filter, as shown in [Figure 17-9](#). There is no design-time validation of the filter query. You must ensure the query is correct.

Figure 17-9 Creating a Sensor Action with a Filter



A new entry is created in the `bpel_process_name_sensorAction.xml` file, as shown in [Example 17-6](#):

Example 17–6 *bpel_process_name_sensorAction.xml* file

```

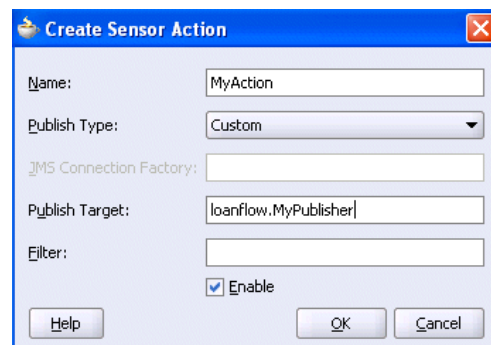
<action name="BigMoneyBAMAction"
  enabled='true'
  filter="boolean(/s:actionData/s:payload
                /s:variableData/s:data
                /autoloan:loanAmount > 100000)"
  publishType="JMSQueue"
  publishTarget="jms/bigMoneyQueue">
  <sensorName>LoanApplicationSensor</sensorName>
  <property name="JMSConnectionFactory">
    weblogic.jms.ConnectionFactory
  </property>
</action>

```

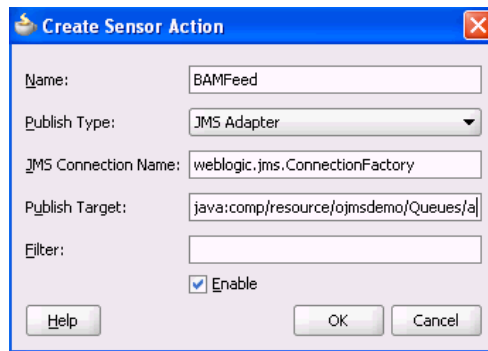
Notes:

- You must specify all the namespaces that are required to configure an action filter in the *bpel_process_name_sensorAction.xml* configuration file. For example, assume you have a customer XML-schema element with namespace "http://myCustomer" and you want to create a filter on the customer age element. Therefore, you must manually declare the namespace for "http://myCustomer" in the file before you can use it in your filter. Otherwise, it is not possible to create a valid query. Add `xmlns:ns1="http://myCustomer"` in the attribute declaration part of the file. You can then use `....//ns1:customer/ns1:age/...` in your query.
- You must specify the filter as a boolean XPath expression.

If you have special requirements for a sensor action that cannot be accomplished by using the built-in publish types (database, JMS queue, JMS topic, and JMS Adapter), then you can create a sensor action with the custom publish type, as shown in [Figure 17–10](#). The name in the **Publish Target** field denotes a fully qualified Java class name that must be implemented. For more information, see [Section 17.2.5, "How to Create a Custom Data Publisher."](#)

Figure 17–10 Using the Custom Publish Type**17.2.4 How to Publish to Remote Topics and Queues**

The JMS queue and JMS topic publish types only publish to local JMS destinations. If you want to publish sensor data to remote topics and queues, use the JMS adapter publish type, as shown in [Figure 17–11](#).

Figure 17–11 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 `weblogic-ra.xml` file, which is updated through editing in the Oracle WebLogic Server Administration Console. Each JMS connection factory (pool) entry created in this console corresponds to one JNDI entry in `weblogic-ra.xml`. Update the Sensor Actions dialog with the chosen JNDI name selected during the creation of the JMS connection factory (pool).

For more information about the JMS adapter, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

17.2.5 How to Create a Custom Data Publisher

To create a custom data publisher, perform the following steps:

To create a custom data publisher:

1. In the Application Navigator, double-click the BPEL project.

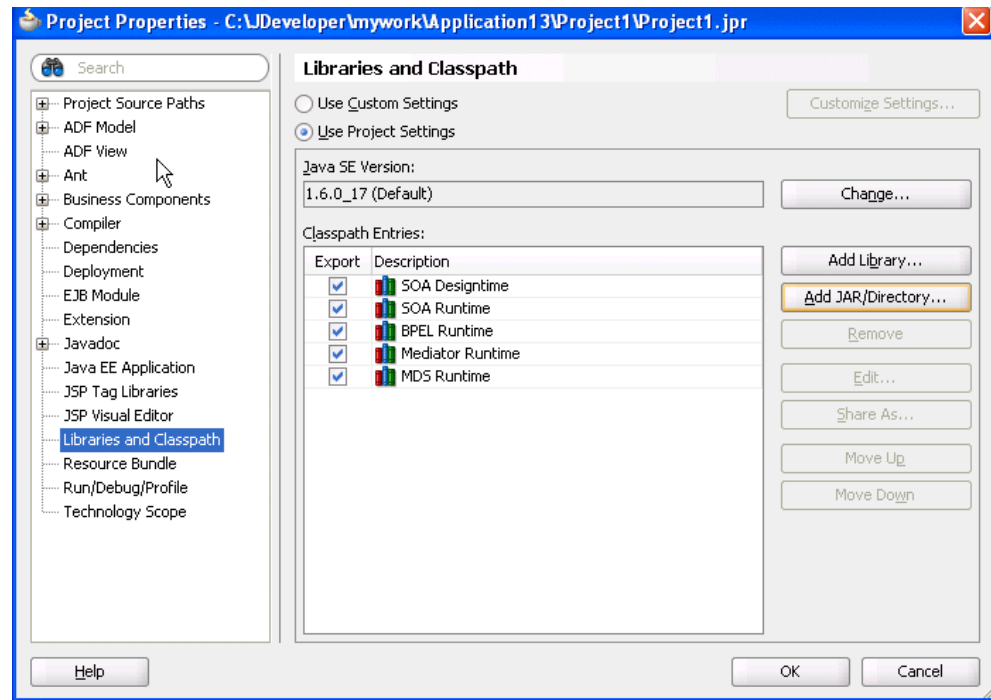
The Project Properties dialog appears.

2. Click **Libraries and Classpath**.

3. Browse and select the following:

`SOA_ORACLE_HOME\lib\java\shared\oracle.soainfra.common\11.1.1\orabpel.jar`

Figure 17–12 provides details.

Figure 17–12 Project Properties Dialog

4. Create a new Java class.

The package and class name must match the publish target name of the sensor action.

5. Implement the `com.oracle.bpel.sensor.DataPublisher` interface.

This updates the source file and fills in the methods and import statements of the **DataPublisher** interface.

6. Using the Oracle JDeveloper editor, implement the publish method of the **DataPublisher** interface, as shown in the sample custom data publisher class in [Figure 17–13](#).

Figure 17–13 Custom Data Publisher Class

```

MyPublisher.java
package loanflow;

import com.oracle.bpel.sensor.DataPublisher;

import com.oracle.bpel.sensor.schemas.ITHeaderInfo;
import com.oracle.bpel.sensor.schemas.ITSensorAction;
import com.oracle.bpel.sensor.schemas.ITSensorActionData;
import com.oracle.bpel.sensor.schemas.ITSensorData;

import org.w3c.dom.Element;

public class MyPublisher implements DataPublisher
{
    public MyPublisher()
    {
    }

    public void publish(ITSensorAction action,
                       ITSensorActionData actionData,
                       Element xml) throws Exception
    {
        ITHeaderInfo header = actionData.getHeader();
        ITSensorData data = actionData.getPayload();

        // Print information on who fired the sensor
        System.out.println("Sensor " + header.getSensor().getSensorName() +
                           " fired for BPEL process " + header.getProcessName());

        // Print the sensor data to the console
        System.out.println("Sensor data: " + xml.toString());
    }
}

```

7. Ensure that the class compiles successfully.

The next time that you deploy the BPEL process, the Java class is added to the SOA archive (SAR) and deployed.

Note: Ensure that additional Java libraries needed to implement the data publisher are in the class path.

Oracle BPEL Process Manager can execute multiple process instances simultaneously, so ensure that the code in your data publisher is thread safe, or add appropriate synchronization blocks. To guarantee high throughput, do not use shared data objects that require synchronization.

17.2.6 How to Register the Sensors and Sensor Actions in composite.xml

Oracle JDeveloper automatically updates the `composite.xml` file to include appropriate properties for sensors and sensor actions, as shown in [Example 17–7](#):

Example 17–7 composite.xml File

```

<composite name="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 [Example 17-7](#).

17.3 Viewing Sensors and Sensor Action Definitions in Oracle Enterprise Manager Fusion Middleware Control

The Oracle Enterprise Manager Fusion Middleware Control provides support for viewing the metadata of sensors, sensor actions, and the sensor data created as part of the process execution.

For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

Notes:

- For this release, BAM sensor actions are not shown in Oracle Enterprise Manager Fusion Middleware Control.
 - Only sensors with an associated database sensor action are displayed in Oracle Enterprise Manager Fusion Middleware Control. Sensors associated with a JMS queue, JMS topic, remote JMS, or custom sensor action are not displayed.
-

Part III

Using the Oracle Mediator Service Component

This part describes the components that comprise the Oracle Mediator service component.

This part contains the following chapters:

- [Chapter 18, "Getting Started with Oracle Mediator"](#)
- [Chapter 19, "Creating Oracle Mediator Routing Rules"](#)
- [Chapter 20, "Working with Multiple Part Messages in Oracle Mediator"](#)
- [Chapter 21, "Using Oracle Mediator Error Handling"](#)
- [Chapter 22, "Resequencing in Oracle Mediator"](#)
- [Chapter 23, "Understanding Message Exchange Patterns of an Oracle Mediator"](#)

Getting Started with Oracle Mediator

This chapter provides an overview of Oracle Mediator and also describes how to create an Oracle Mediator service component.

This chapter includes the following sections:

- [Section 18.1, "Introduction to Oracle Mediator"](#)
- [Section 18.2, "Introduction to the Mediator Editor Environment"](#)
- [Section 18.4, "Configuring the Oracle Mediator Interface Definition"](#)
- [Section 18.5, "Generating a WSDL File"](#)
- [Section 18.6, "Specifying Operation or Event Subscription Properties"](#)
- [Section 18.7, "Modifying an Oracle Mediator Service Component"](#)

18.1 Introduction to Oracle Mediator

Oracle Mediator is a service component of the Oracle SOA Suite that provides mediation capabilities such as selective routing, transformation, and validation capabilities, along with various message exchange patterns, such as synchronous, asynchronous, and event publishing or subscriptions.

Oracle Mediator provides a lightweight framework to mediate between various components within a composite application. Oracle Mediator converts data to facilitate communication between different interfaces exposed by different components that are wired to build a SOA composite application. For example, Oracle Mediator can accept data contained in a text file from an application or service, transform it into a format appropriate for updating a database that serves as a customer repository, and then route and deliver the data to that database.

Oracle Mediator facilitates integration between events and services, where service invocations and events can be mixed and matched. You can use an Oracle Mediator service component to consume a business event or receive a service invocation. An Oracle Mediator service component can evaluate routing rules, perform transformations, validate, and either invoke another service or raise another business event. You can use an Oracle Mediator service component to handle returned responses, callbacks, faults, and timeouts.

Oracle Mediator provides the following features:

- Content-Based and Header-Based Routing

Oracle Mediator provides support for setting rules based on message payload or message headers. You can select elements or attributes from the message payload or the message header and, based on the values in those elements or attributes,

you can specify an action. For example, Oracle Mediator receives a file from an application or service containing data about new customers. Based on the country mentioned in the customer's address, you can route and deliver data to the database storing data for that particular country. Similarly, you can route a message based on the message header.

For more information about header-based routing, see [Section 19.2.2.11, "How to Access Headers for Filters and Assignments."](#)

- Synchronous and Asynchronous Interactions

Oracle Mediator provides support for synchronous and asynchronous request and response interactions. In a synchronous interaction, the client requests a service and then waits for a response to the request. In an asynchronous interaction, the client invokes the service, but does not wait for the response. You can specify a timeout period for an asynchronous interaction and you can specify an action to perform after the timeout period, such as raise an event or start a process.

For more information about synchronous and asynchronous interactions, see [Section 19.2.2.4, "How to Configure Response Messages"](#) and [Chapter 23, "Understanding Message Exchange Patterns of an Oracle Mediator."](#)

- Sequential and Parallel Routing of Messages

A routing rule can be either executed in parallel or sequentially. You can configure the execution type from the **Routing Rules** section of the Mediator Editor.

For more information about sequential and parallel routing of messages, see [Section 19.2.2.3, "How to Specify Sequential or Parallel Execution."](#)

- Transformations

Oracle Mediator supports data transformation from one XML schema to another. This feature enables data interchange among applications using different schemas. For example, you can transform a comma-delimited file to a database table structure.

For more information about transformations, see [Section 19.2.2.8, "How to Create Transformations."](#)

- Validations

Oracle Mediator provides support for validating the incoming message payload using a Schematron or an XSD file. You can specify Schematron files for each inbound message part and Oracle Mediator can execute Schematron file validations for those parts.

For more information about validations, see [Section 19.2.2.12, "How to Use Semantic Validation"](#) and <http://www.schematron.com/>.

- Java Callouts

Oracle Mediator lets you add Java callouts to the routing rules. Java callouts are a way of using of Java code with regular expressions.

For more information about Java callouts, see [Section 19.2.2.13, "How to Use Java Callouts."](#)

- Event Handling

An event is a message sent because an activity occurred in a business environment. Oracle Mediator supports subscribing to business events and raising business events. You can subscribe to a business event that is generated when a situation of interest occurs. For example, you can subscribe to an event that is

generated when a new customer is created and then use this event to start a business process, such as sending a confirmation email. Similarly, you can generate business events when a situation of interest occurs. For example, after a new customer profile is created, you can generate a customer-created event.

For more information about event handling, see [Chapter 38, "Using Business Events and the Event Delivery Network."](#)

- **Dynamic Routing**

Dynamic routing separates the control logic of a process from the execution of the process. The control logic determines the path taken by the process. You can create a dynamic routing rule from the Mediator Editor.

For more information about dynamic routing, see [Section 19.2.3, "How to Create Dynamic Routing Rules."](#)

- **Error Handling**

Oracle Mediator supports both manual error handling and error handling based on a fault policy. A fault policy consists of conditions and actions, where the conditions specify the action to be carried out for a particular error condition.

For more information about error handling, see [Chapter 21, "Using Oracle Mediator Error Handling."](#)

- **Echo**

Oracle Mediator supports echoing source messages back to the initial caller after any transformations, validations, assignments, or sequencing operations are performed.

For more information about Oracle Mediator echo support, see ["To echo a service:"](#) of [Section 19.2.2.1, "How to Specify Oracle Mediator Services or Events."](#)

- **Multiple Part Messages**

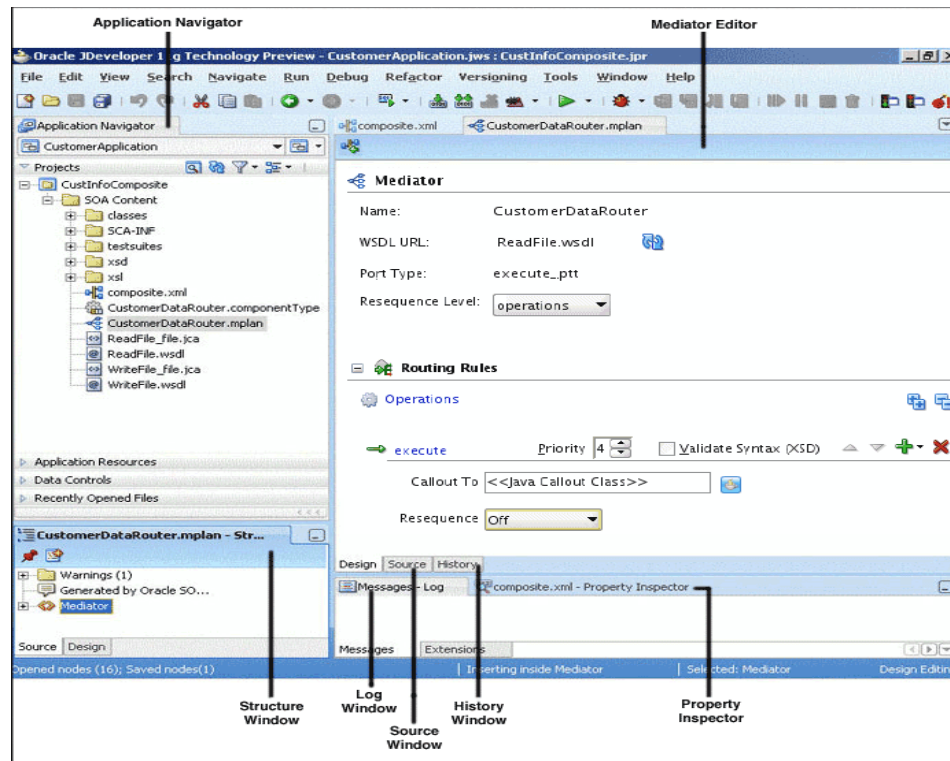
Oracle Mediator can process messages consisting of multiple parts. Some Remote Procedure Call (RPC) web services contain multiple parts in the SOAP message.

For more information about multiple part message support, see [Chapter 20, "Working with Multiple Part Messages in Oracle Mediator."](#)

18.2 Introduction to the Mediator Editor Environment

You can create an Oracle Mediator service component in a SOA composite application of Oracle JDeveloper and then configure it using the Mediator Editor. To display the Mediator Editor, double-click the Oracle Mediator service component in the SOA Composite Editor. For information about the SOA Composite Editor, see [Chapter 2, "Developing SOA Composite Applications with Oracle SOA Suite."](#)

[Figure 18–1](#) shows the Mediator Editor along with the Application Navigator, Structure, and Messages windows.

Figure 18–1 Mediator Editor Window

Each section of the view shown in [Figure 18–1](#) lets you perform specific design and deployment tasks. The sections in this view include the following:

- **Application Navigator**

The Application Navigator, shown in the upper left section of [Figure 18–1](#), displays the Oracle Mediator file structure. These files appear under the SOA Content folder of the project where you created an Oracle Mediator.

A SOA composite application consists of the following Oracle Mediator files:

- `composite.xml`: This file describes the entire SOA composite application. For information about the `composite.xml` file, see [Chapter 2, "Developing SOA Composite Applications with Oracle SOA Suite."](#)
- `.componentType`: This file describes the services and references for a service component.
- `.mplan`: This file contains Oracle Mediator metadata.
- `.wsdl`: The Web Services Description Language (WSDL) file specifies how other services call an Oracle Mediator. A WSDL file defines the input and output messages and operations of an Oracle Mediator.

- **Mediator Editor**

The Mediator Editor, shown in the middle of [Figure 18–1](#), provides a visual view of the Oracle Mediator component. This view appears when you perform one of the following actions:

- Double-click an **Oracle Mediator** icon in the SOA Composite Editor.
- Double-click the `.mplan` file for the Oracle Mediator in the Application Navigator.

- Source View

The Source view displays the source code of an Oracle Mediator. Click **Source** at the bottom of the Mediator Editor shown in [Figure 18–1](#) to view the source code. The code in Source view is immediately updated to reflect any changes to an Oracle Mediator.

[Example 18–1](#) shows sample Oracle Mediator source code:

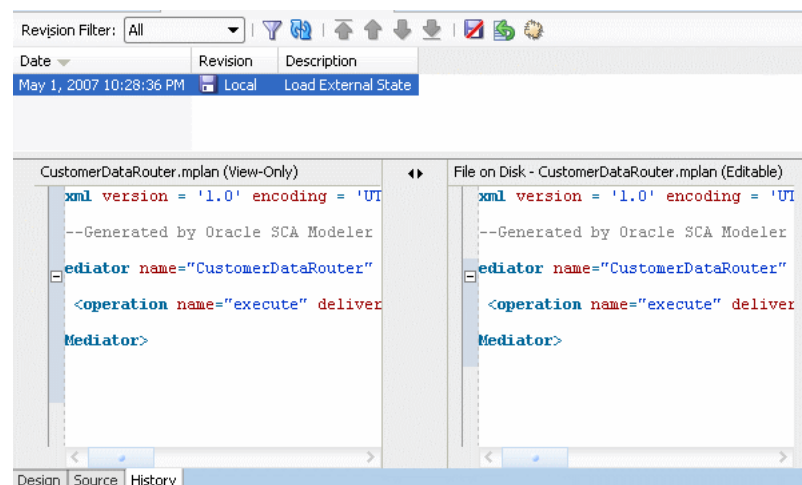
Example 18–1 Oracle Mediator Source Code

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

- History Window

The History window displays history information about the Oracle Mediator file, including a revision history and side-by-side comparisons of read-only and editable versions of a file. Click **History** at the bottom of the Design window shown in [Figure 18–1](#) to open the History window. [Figure 18–2](#) shows the History view for an Oracle Mediator file.

Figure 18–2 History Window



- Property Inspector

The Property Inspector, shown at the bottom of [Figure 18–1](#), displays details about Oracle Mediator properties.

- Structure Window

The Structure Window, shown in the lower left section of [Figure 18–1](#), displays a structural view of the data of an Oracle Mediator.

- Log Window

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

18.3 Creating an Oracle Mediator

You can create an Oracle Mediator in multiple ways, depending on where you are in your application development process.

18.3.1 How to Create an Oracle Mediator

You can create an Oracle Mediator in a SOA composite application of Oracle JDeveloper using one of the following methods:

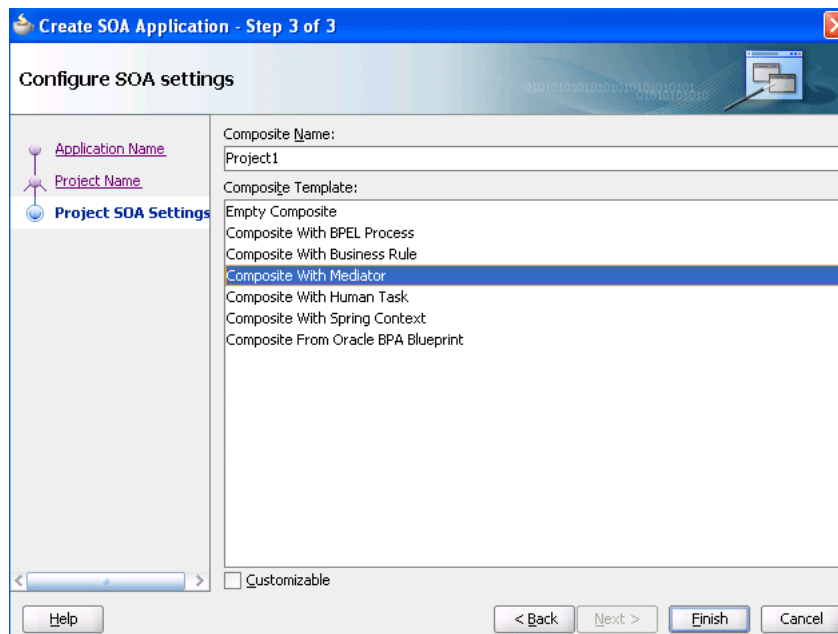
- When you create a composite application
- From within an existing composite application
- When you create a project
- From within an existing project

When you create an Oracle Mediator using any of these methods, the Create Mediator dialog appears so you can name the Oracle Mediator component and select a template for the interface.

To create an Oracle Mediator when creating a composite application:

1. Create and Name the SOA application and project using the Create SOA Application wizard.
2. When you reach the Configure SOA Settings page, select **Composite with Mediator** in the Composite Template list, as shown in [Figure 18–3](#).

Figure 18–3 Composite with Oracle Mediator Selection in Create SOA Project Wizard



3. Click **Finish**.

The Create Mediator dialog appears.

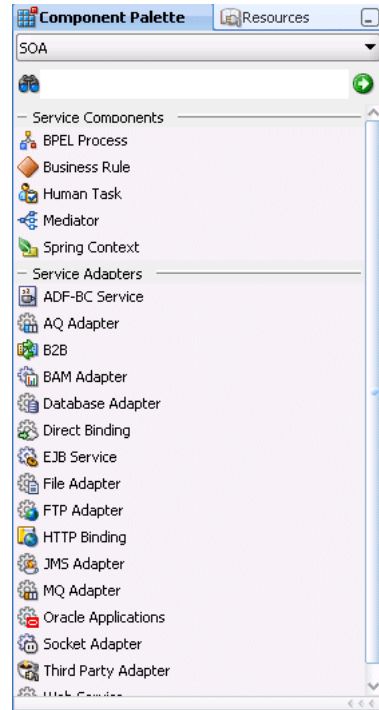
4. Configure the Oracle Mediator interface, as described in [Section 18.4, "Configuring the Oracle Mediator Interface Definition"](#).

To create an Oracle Mediator within a composite application:

1. Open the composite application to which you are adding an Oracle Mediator in the SOA Composite Editor.
2. Drag and drop an Oracle Mediator from the Component Palette (shown in [Figure 18-4](#)) to the **Components** section of the editor.

Tip: The Component Palette is to the right of the SOA Composite Editor.

Figure 18-4 *Component Palette with an Oracle Mediator Service Component*



The Create Mediator dialog appears.

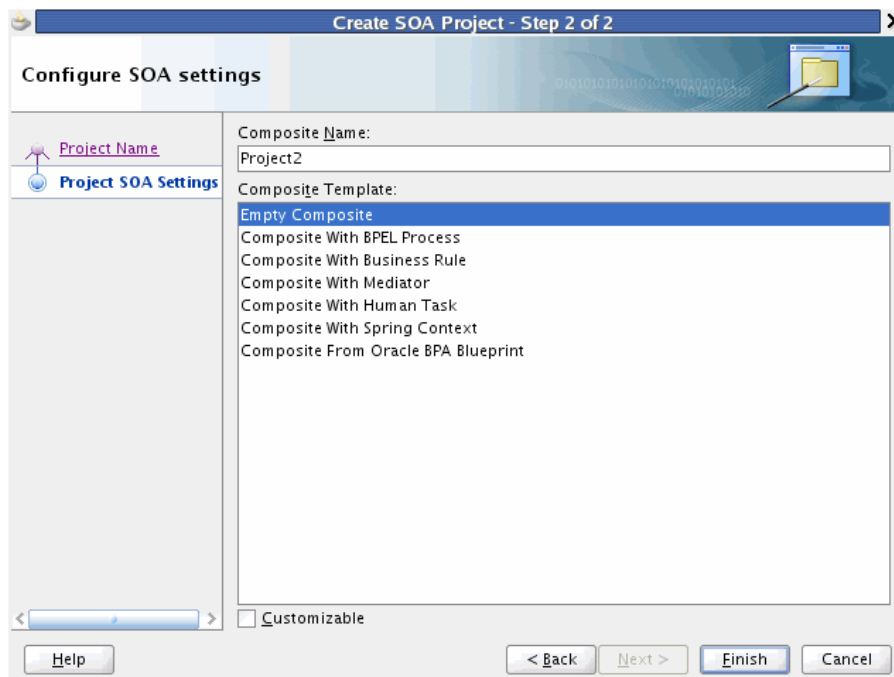
3. Configure the Oracle Mediator interface, as described in [Section 18.4, "Configuring the Oracle Mediator Interface Definition"](#).

To create an Oracle Mediator when creating a new project:

1. Using the New Gallery wizard, create and name a new SOA project in the SOA Tier category.

Tip: The New Gallery wizard appears when you select **New Project** from the Application menu to the right of the application name in the Application Navigator. You can also right-click in the Application Navigator and select **New**.

2. On the Configure SOA Settings page of the New Gallery dialog, select **Composite With Mediator** from the **Composite Template** list, shown in [Figure 18-5](#).

Figure 18–5 Create SOA Project Wizard with Composite With Mediator Template Shown

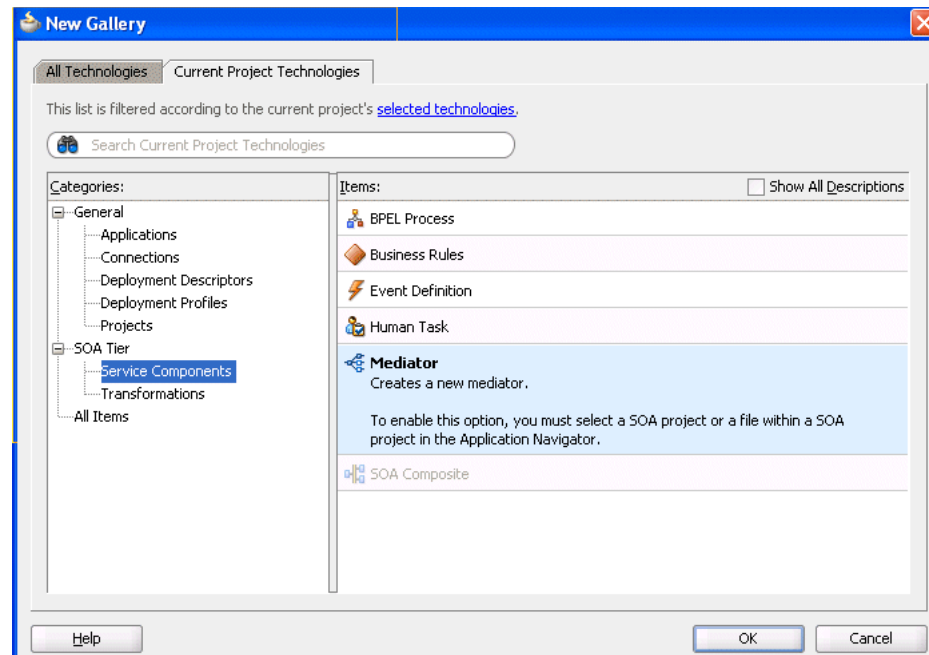
3. Click **Finish**.

The Create Mediator dialog appears.

4. Configure the Oracle Mediator interface, as described in [Section 18.4, "Configuring the Oracle Mediator Interface Definition"](#).

To create an Oracle Mediator within an existing project:

1. In the Application Navigator, select the project to which you want to add an Oracle Mediator.
2. Right-click in the navigator pane and select **New**.
3. Under **Categories**, select **Service Components**, and then select **Mediator** from the **Items** list, as shown in [Figure 18–6](#).

Figure 18–6 New Gallery Dialog with Oracle Mediator Service Component

4. Click **OK**.
The Create Mediator dialog appears.
5. Configure the Oracle Mediator interface, as described in [Section 18.4, "Configuring the Oracle Mediator Interface Definition"](#).

18.4 Configuring the Oracle Mediator Interface Definition

When you create a new Oracle Mediator, you can specify an interface template that generates a basic set of default files in the Oracle Mediator project. These files provide a framework from which you can design and configure the Oracle Mediator. You can create an Oracle Mediator with the following interface options:

- Oracle Mediator with no interface definition
- Oracle Mediator with the interface defined by a WSDL file
- Oracle Mediator with a one-way interface
- Oracle Mediator with a synchronous interface
- Oracle Mediator with an asynchronous interface
- Oracle Mediator that subscribes to events

18.4.1 Creating an Oracle Mediator Without an Interface Definition

You can create an empty Oracle Mediator with no interface definition. This process does not create a WSDL file, and it provides you with the flexibility to create the SOA components in the order you want. After you create an Oracle Mediator without an interface definition, you must create a service or an event that starts the Oracle Mediator.

18.4.1.1 How to Create an Oracle Mediator Without an Interface Definition

The **Define Interface Later** template in the Create Mediator dialog creates an Oracle Mediator with no interface definition.

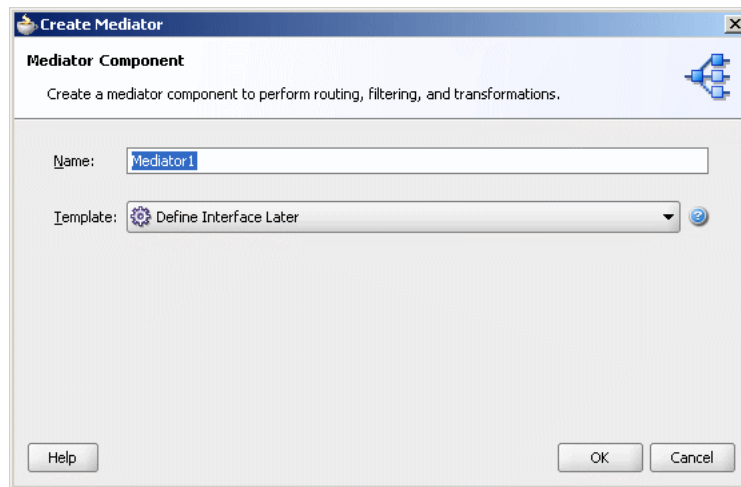
To create an Oracle Mediator without an interface definition:

1. Create an Oracle Mediator using one of the methods described in [Section 18.3, "Creating an Oracle Mediator"](#).

The Create Mediator dialog appears.

2. In the **Name** field, enter a name for the Oracle Mediator service component.
3. From the **Template** list, select **Define Interface Later** and click **OK**.

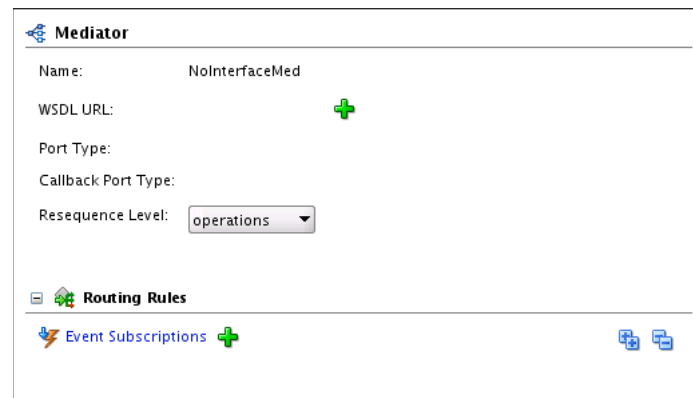
Figure 18–7 Define Interface Later Template Selection on the Create Mediator Dialog



18.4.1.2 What Happens When You Create an Oracle Mediator Without an Interface Definition

The Oracle Mediator files are generated under the specified application and project in the Application Navigator, and the new Oracle Mediator appears in the Mediator Editor in Design view.

The Oracle Mediator has no associated WSDL file, port types, or operations. You must define these as described in the following sections. [Figure 18–8](#) shows how an Oracle Mediator created with no interface definition appears in the Mediator Editor.

Figure 18–8 Oracle Mediator with no Interface Definition in the Mediator Editor

18.4.1.3 How to Define an Interface for an Oracle Mediator

After you create an Oracle Mediator without an interface definition, you must define the interface by subscribing to events or by defining services.

To subscribe to events:

To subscribe to events, the events must be defined in an Event Definition (EDL) file.

1. Open the Oracle Mediator you want to edit in the Mediator Editor.
2. In the **Routing Rules** section, click **Add Event Subscription**.
The Subscribed Events dialog appears.
3. Click **Add**.
The Event Chooser dialog appears.
4. To the right of the **Event Definition File** field, click **Search** and then browse to 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 publisher** field, either leave the default value of **yes** or select **no**.
8. Double-click the **Filter** field to open the Expression Builder and define an expression for filtering the event.
9. Click **OK**.

For more information about the **Consistency**, **Run as publisher**, and **Filter** fields of an event, see [Section 18.4.6, "Creating an Oracle Mediator for an Event Subscription."](#)

To define services:

You can define service for an Oracle Mediator with no interface definition in the following two ways:

- Connect the Oracle Mediator to a service through a wire in the SOA Composite Editor.
- Use the **Define Service** option in the Mediator Editor.

To define services for an Oracle Mediator through a wire:

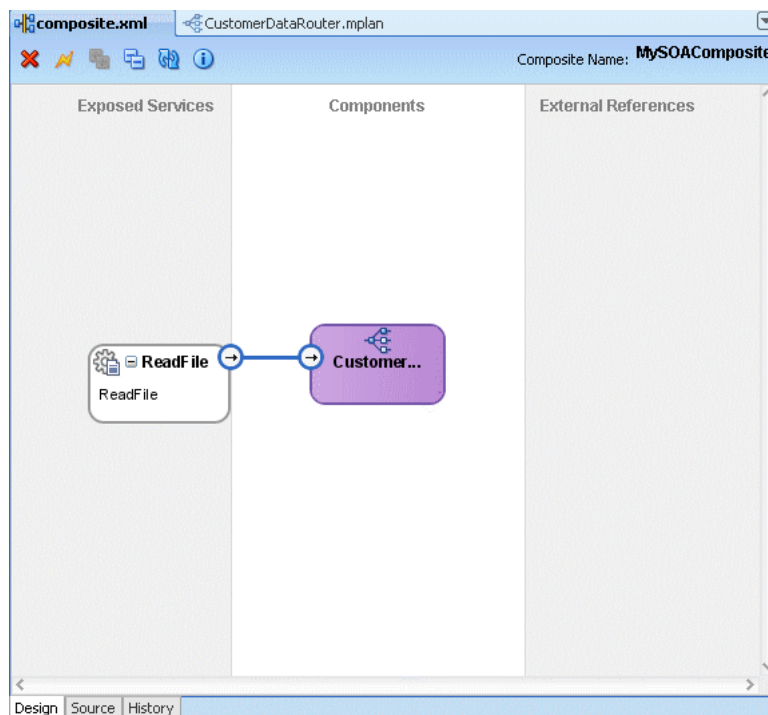
- In the SOA Composite Editor, drag a wire from an Oracle Mediator to a service.

For more information about wires and how to wire a service component to a service, see [Section 2.5.1, "How to Wire a Service and a Service Component."](#)

Note: You can also connect an Oracle Mediator with a defined interface and defined reference to a service through a wire. However, to connect Oracle Mediator to a service, the interface of the Oracle Mediator and of the service must match.

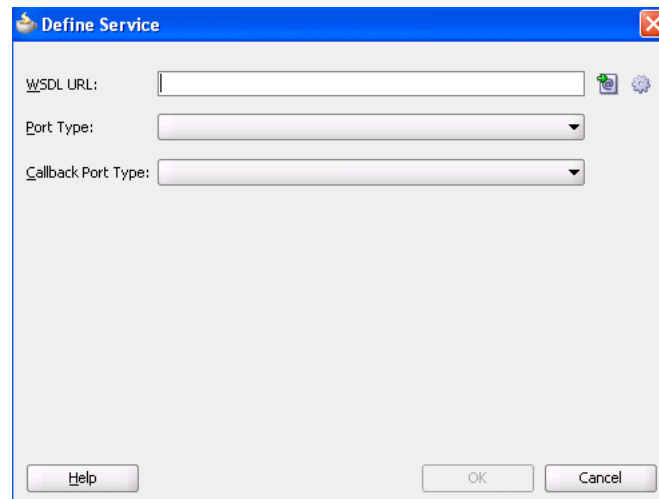
The service for an Oracle Mediator is automatically defined using the WSDL file from the wire source. For example, if you connect the ReadFile service shown in [Figure 18–9](#) to the CustomerDataRouter Oracle Mediator, the CustomerDataRouter Oracle Mediator automatically inherits the service definition of the ReadFile service.

Figure 18–9 Connecting Oracle Mediator to a Service

**To define services for an Oracle Mediator in the Mediator Editor:**

1. Display the Oracle Mediator you want to edit in the Mediator Editor.
2. To the right of the **WSDL URL** field, click **Define Service**.

The Define Service dialog appears, as shown in [Figure 18–10](#).

Figure 18–10 Define Service Dialog

3. Do one of the following:
 - To use an existing WSDL file, click **Find existing WSDLs** to the right of the **WSDL URL** field.
 - To create a WSDL file, click **Generate WSDL from schema(s)** to the right of the **WSDL URL** field.

For information about how to generate a WSDL file, see [Section 18.5, "Generating a WSDL File."](#)

4. From the **Port Type** list, select a port.
5. From the **Callback Port Type** list, select a port for the response message in an asynchronous interaction.
6. Click **OK**.

18.4.2 Creating an Oracle Mediator Based on a WSDL File

When you create an Oracle Mediator, you can base the interface definition on a WSDL file, which describes the interfaces of an Oracle Mediator, such as port types, operations, services, and schemas.

18.4.2.1 How to Create an Oracle Mediator Based on a WSDL File

The **Interface Definition from WSDL** template on the Create Mediator dialog box creates an Oracle Mediator based on a WSDL file that has already been created or that can be generated from a schema.

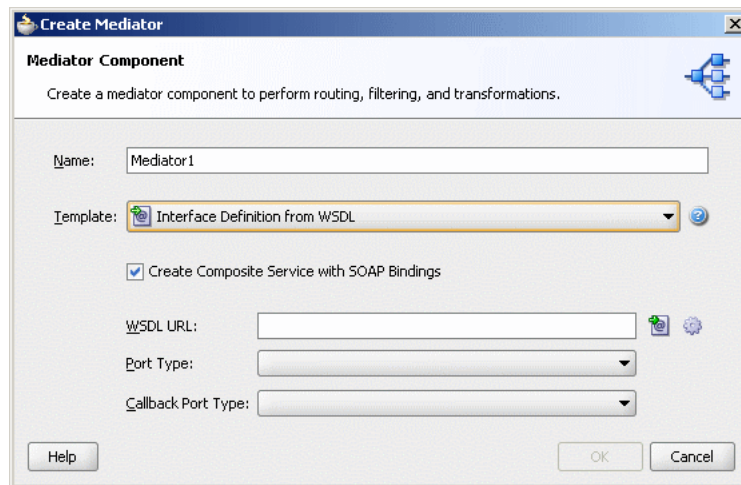
To create an Oracle Mediator based on a WSDL file:

1. Create an Oracle Mediator as described in [Section 18.3, "Creating an Oracle Mediator"](#).

The Create Mediator dialog appears.

2. In the **Name** field, enter a name for the Oracle Mediator service component.
3. From the **Template** list, select **Interface Definition from WSDL**, as shown in [Figure 18–11](#).

Figure 18–11 Interface Definition from WSDL Template Selection on the Create Mediator Dialog



4. If you do not want to create an exposed service with SOAP bindings that is automatically connected to your Oracle Mediator, deselect the **Create Composite Service with SOAP Bindings** option.
5. In the **WSDL URL** field, do one of the following:
 - To use an existing WSDL file, enter the name of the file or click **Find existing WSDL files** to browse for the file.
 - To create a new WSDL file, click **Generate WSDL from schema(s)**.

For more information about these options, see [Section 18.5, "Generating a WSDL File."](#)

6. From the **Port Type** list, select a port.
This parses the WSDL file that you specify in the **WSDL URL** field to display the list of port types.
7. From the **Callback Port Type** list, select a callback port.
A callback port is the one to which the response message is sent in an asynchronous communication.
8. Click **OK**.

18.4.2.2 What Happens When You Create an Oracle Mediator from a WSDL File

The Oracle Mediator files are generated under the specified application and project in the Application Navigator, and the new Oracle Mediator appears in the Mediator Editor in Design view. If the WSDL file you specify is located in a different directory from the project files, the file and its associated schema files are copied to the Oracle Mediator project.

The appearance and source code of the Oracle Mediator varies depending on the name of the WSDL file and the port types and operations defined by the WSDL file.

[Figure 18–12](#) shows a sample Oracle Mediator created from a WSDL file.

Figure 18–12 Oracle Mediator from WSDL in the Mediator Editor

The screenshot shows the 'Mediator' configuration window. The 'Name' field is 'MediatorFromWSDL'. The 'WSDL URL' is 'BETWMediator.wsdl'. The 'Port Type' is 'execute_ptt'. The 'Resequencing Level' is set to 'operations'. Below this, the 'Routing Rules' section is expanded, showing an 'Operations' tab. Under 'Operations', there is an 'execute' operation with a 'Priority' of 4. The 'Callout To' field is set to '<<Java Callout Class>>'. The 'Resequencing' checkbox is 'Off'.

18.4.3 Creating an Oracle Mediator With a One-Way Interface Definition

In a one-way interaction, the client sends a message to a service and the service does not need to reply.

18.4.3.1 How to Create an Oracle Mediator with a One-Way Interface Definition

The **One-Way Interface** template in the Create Mediator dialog creates an Oracle Mediator for a one-way interaction.

To create an Oracle Mediator with a one-way interface definition:

1. Create an Oracle Mediator as described in [Section 18.3, "Creating an Oracle Mediator"](#).

The Create Mediator dialog appears.

2. In the **Name** field, enter a name for the Oracle Mediator service component.
3. From the **Template** list, select **One-Way Interface**, as shown in [Figure 18–13](#).

Figure 18–13 One-Way Interface Template Selection on the Create Mediator Dialog

The screenshot shows the 'Create Mediator' dialog box. The 'Mediator Component' section is active. The 'Name' field contains 'Mediator1'. The 'Template' dropdown menu is set to 'One-Way Interface'. The 'Create Composite Service with SOAP Bindings' checkbox is checked. The 'Input' field contains the XML schema path '{http://xmlns.oracle.com/singleString}singleString'. The 'Help', 'OK', and 'Cancel' buttons are at the bottom.

4. If you do not want to create an exposed service with SOAP bindings that is automatically connected to your Oracle Mediator service component, deselect the **Create Composite Service with SOAP Bindings** option.
5. To the right of the **Input** field, click **Search** to select a schema element for the input message.

By default, the **singleString** schema element is selected for the input message.

Note: You can use any XSD schema to specify the format of the input document that Oracle Mediator processes. Here is a sample schema:

```
<xsd:schema attributeFormDefault="qualified"
  elementFormDefault="qualified"
  targetNamespace="http://samples.otn.com/helloworld"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://samples.otn.com/helloworld">
  <include namespace="http://samples.otn.com/helloworld"
    schemaLocation="helloworld.xsd" />
  <xsd:element name="name1" type="xsd:string" />
  <xsd:element name="result1" type="xsd:string"/>
</xsd:schema>
```

6. Click **OK**.

18.4.3.2 What Happens When You Create an Oracle Mediator with a One-Way Interface Definition

The Oracle Mediator files that define a one-way interaction are generated under the specified application and project in the Application Navigator, and the new Oracle Mediator appears in the Mediator Editor in Design view. A WSDL file is also generated with the same name as the Oracle Mediator.

Figure 18–14 shows how an Oracle Mediator created with a one-way interface appears in the Mediator Editor. The arrow to the left of the **execute** operation in Figure 18–16 represents a one-way operation.

Figure 18–14 One-Way Interface Oracle Mediator in the Mediator Editor



18.4.4 Creating an Oracle Mediator with a Synchronous Interface Definition

Oracle Mediator supports synchronous request-response interactions. In a synchronous interaction, a client sends a request to a service and receives an immediate response. The client does not proceed further until the response arrives.

18.4.4.1 How to Create an Oracle Mediator with a Synchronous Interface Definition

The **Synchronous Interface** template in the Create Mediator dialog creates an Oracle Mediator for a synchronous interaction.

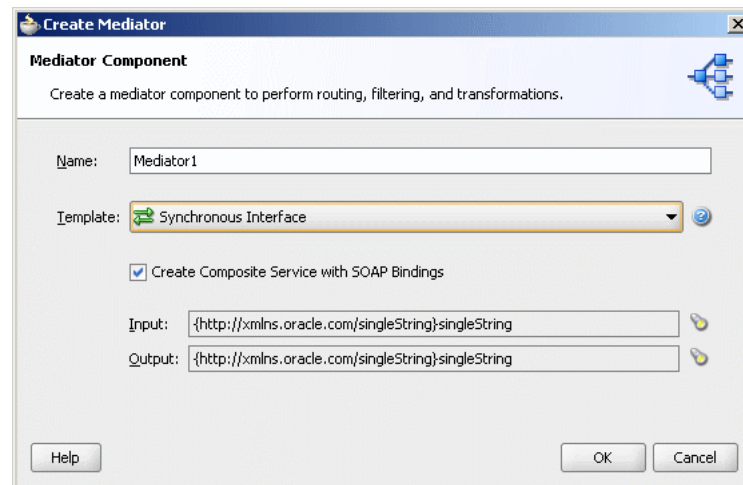
To create an Oracle Mediator with a synchronous interface definition:

1. Create an Oracle Mediator as described in [Section 18.3, "Creating an Oracle Mediator"](#).

The Create Mediator dialog appears.

2. In the **Name** field, enter a name for the Oracle Mediator.
3. From the **Template** list, select **Synchronous Interface**, as shown in [Figure 18–15](#).

Figure 18–15 Synchronous Interface Template Selection on the Create Mediator Dialog



4. If you do not want to create an exposed service with SOAP bindings that is automatically connected to your Oracle Mediator, deselect the **Create Composite Service with SOAP Bindings** option.
5. To the right of the **Input** field, click **Search** to select a schema element for the input message.
By default, the **singleString** schema element is selected for the input message.
6. Click **Search** to the right of the **Output** field to select a schema element for the output message.
By default, the **singleString** schema element is selected for the output message.
7. Click **OK**.

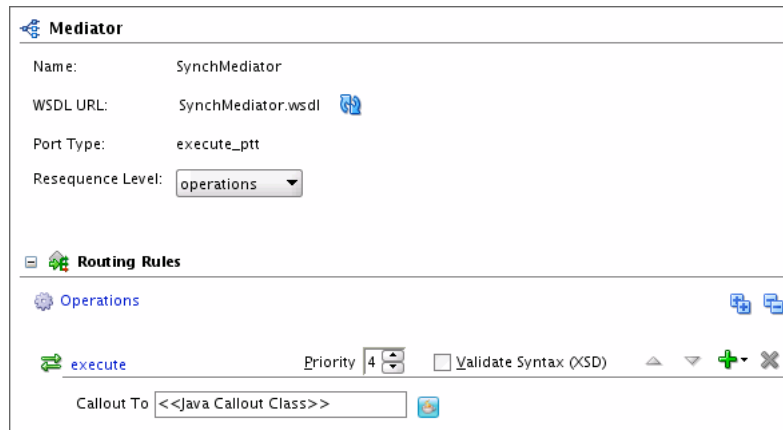
18.4.4.2 What Happens When You Create an Oracle Mediator with a Synchronous Interface Definition

The Oracle Mediator files that define a synchronous interaction are generated under the specified application and project in the Application Navigator, and the new Oracle Mediator appears in the Mediator Editor in Design view. A WSDL file is also generated with the same name as the Oracle Mediator.

In a synchronous interaction, only one port is defined because the response is sent to the same port as the request. [Figure 18–16](#) shows how an Oracle Mediator created with

a synchronous interface appears in the Mediator Editor. The arrows to the left of the **execute** operation in [Figure 18–16](#) represent a synchronous operation.

Figure 18–16 Synchronous Oracle Mediator Component in the Mediator Editor



18.4.5 Creating an Oracle Mediator with an Asynchronous Interface Definition

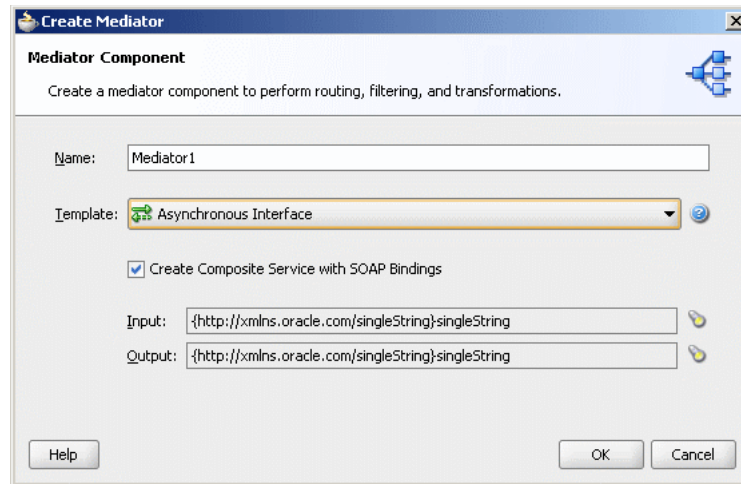
Oracle Mediator supports asynchronous request-response interactions. In an asynchronous interaction, a client sends a request to a service, but does not block and wait for a reply.

18.4.5.1 How to Create an Oracle Mediator with an Asynchronous Interface Definition

The **Asynchronous Interface** template in the Create Mediator dialog creates an Oracle Mediator for asynchronous interaction.

To create an Oracle Mediator with an asynchronous interface definition:

1. Create an Oracle Mediator as described in [Section 18.3, "Creating an Oracle Mediator"](#).
The Create Mediator dialog appears.
2. In the **Name** field, enter a name for the Oracle Mediator.
3. From the **Template** list, select **Asynchronous Interface**, as shown in [Figure 18–17](#).

Figure 18–17 Asynchronous Interface Template Selection on the Create Mediator Dialog

4. If you do not want to create an exposed service with SOAP bindings that is automatically connected to your Oracle Mediator service component, deselect the **Create Composite Service with SOAP Bindings** option.
5. To the right of the **Input** field, click **Search** to select a schema element for the input message.
By default, the **singleString** schema element is selected for the input message.
6. To the right of the **Output** field, click **Search** to select a schema element for the output message.
By default, the **singleString** schema element is selected for the output message.
7. Click **OK**.

18.4.5.2 What Happens When You Create an Oracle Mediator with an Asynchronous Interface Definition

The Oracle Mediator files that define an asynchronous interaction are generated under the specified application and project in the Application Navigator, and the new Oracle Mediator appears in the Mediator Editor in Design view. A WSDL file is also generated with the same name as the Oracle Mediator.

Figure 18–18 shows how an Oracle Mediator created with an asynchronous interface appears in the Mediator Editor. The **Port Type** field displays the port on which the request message is sent. The **Callback Port Type** field displays the port to which the response is sent. The arrows to the left of the **execute** operation in Figure 18–18 represent an asynchronous operation.

Figure 18–18 Asynchronous Oracle Mediator in the Mediator Editor

Mediator

Name: AsynchMediator

WSDL URL: AsynchMediator.wsdl

Port Type: execute_ptt

Callback Port Type: callback_ptt

Resequencing Level: operations

Routing Rules

Operations

execute Priority 4 ☐ Validate Syntax (XSD)

Callout To: <<Java Callout Class>>

Resequencing: Off

18.4.6 Creating an Oracle Mediator for an Event Subscription

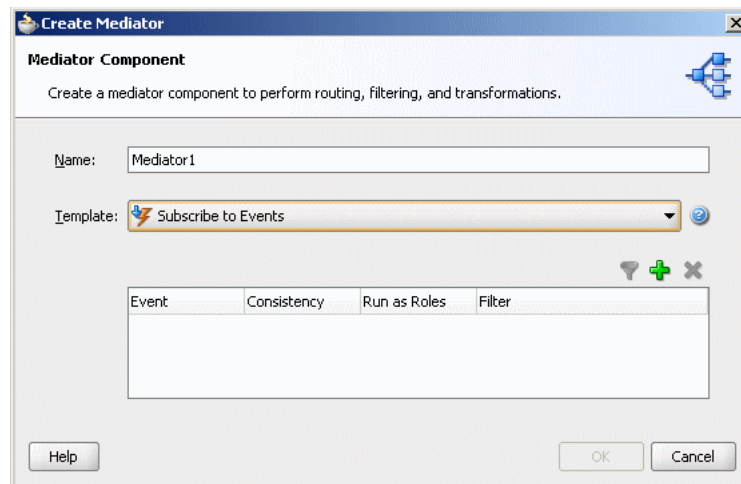
You can create an Oracle Mediator for subscribing to a business event that is generated when a situation of interest occurs. A business event consists of message data sent as the result of an occurrence in a business environment. For information about business events, see [Chapter 38, "Using Business Events and the Event Delivery Network."](#)

18.4.6.1 How to Create an Oracle Mediator for an Event Subscription

The **Subscribe to Events** template in the Create Mediator dialog creates an Oracle Mediator that subscribes to events. To subscribe to events, the events must be defined in an Event Definition (EDL) file.

To create an Oracle Mediator for an event subscription:

1. Create an Oracle Mediator as described in [Section 18.3, "Creating an Oracle Mediator"](#).
The Create Mediator dialog appears.
2. In the **Name** field, enter a name for the Oracle Mediator service component.
3. From the **Template** list, select **Subscribe to Events**, as shown in [Figure 18–19](#).

Figure 18–19 Subscribe to Events Template Selection in Create Mediator Dialog**4. Click Add.**

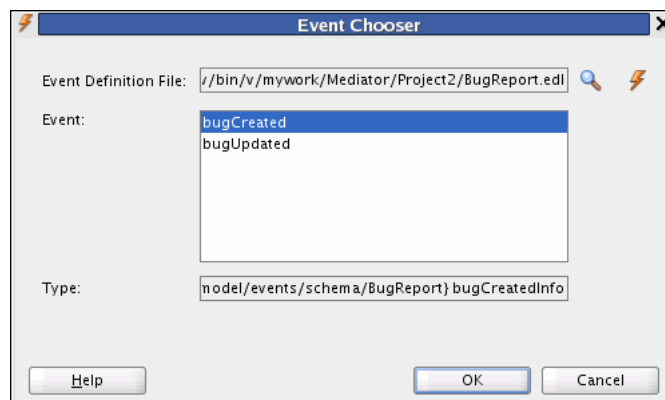
The Event Chooser dialog appears.

5. To the right of the **Event Definition field, click Search.**

The SOA Resource Browser dialog appears.

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

The **Event** field is populated with the events described in the .edl file that you selected. For more information about creating .edl files, see [Chapter 38, "Using Business Events and the Event Delivery Network."](#)

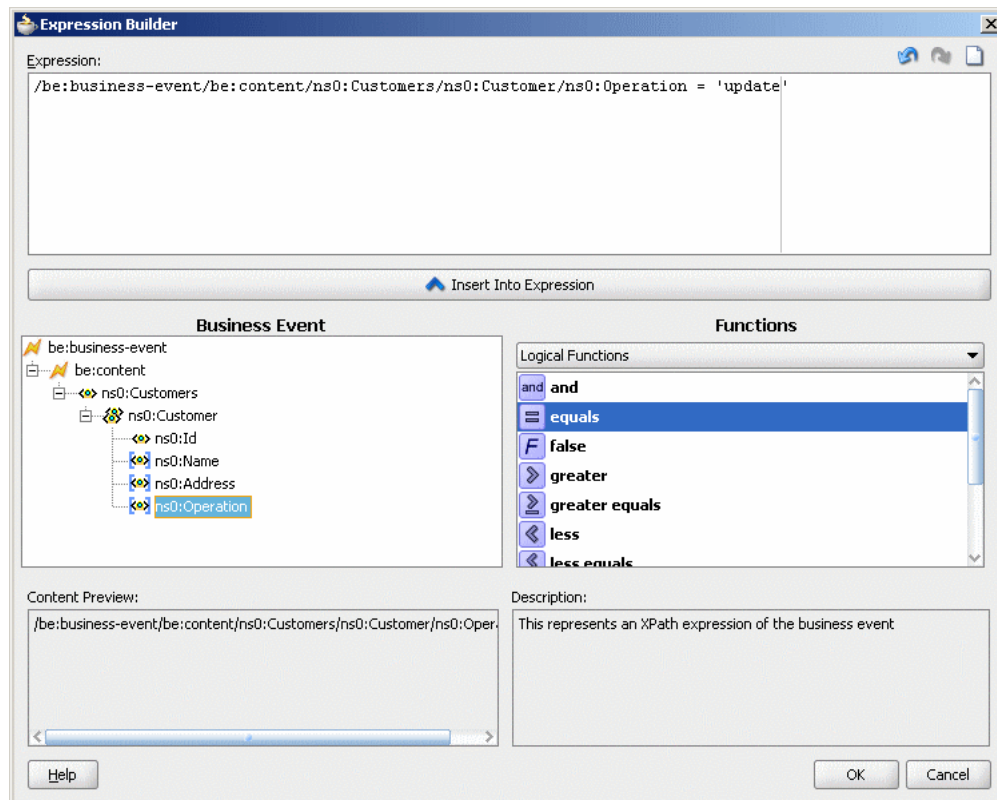
7. Select one or more events in the **Event field, as shown in Figure 18–20, and click OK.****Figure 18–20 Event Chooser Dialog****8. Select a level of delivery consistency for the event.**

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

- **immediate:** Events are delivered on the same thread and on the same transaction as the caller.
9. In the **Run as publisher** field, select whether to run the event subscription under the security of the event publisher.
By default, event subscription run under the security of the event 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 appears.
 11. In the **Expression** field, enter an XPath expression and click **OK**.

Figure 18–21 shows a sample Expression Builder dialog.

Figure 18–21 Business Event Filter



The expression you created appears in the **Filter** column of the Create Mediator dialog.

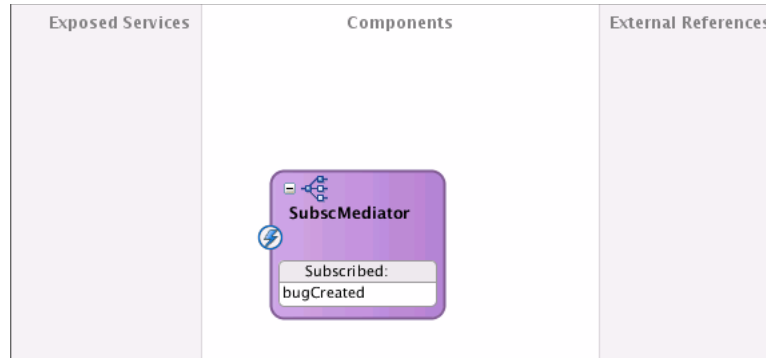
12. Click **OK**.

18.4.6.2 What Happens When You Create an Oracle Mediator for an Event Subscription

The Oracle Mediator files that define an event subscription interaction are generated under the specified application and project in the Application Navigator, and the new Oracle Mediator appears in the Mediator Editor in Design view. A WSDL file is also generated with the same name as the Oracle Mediator.

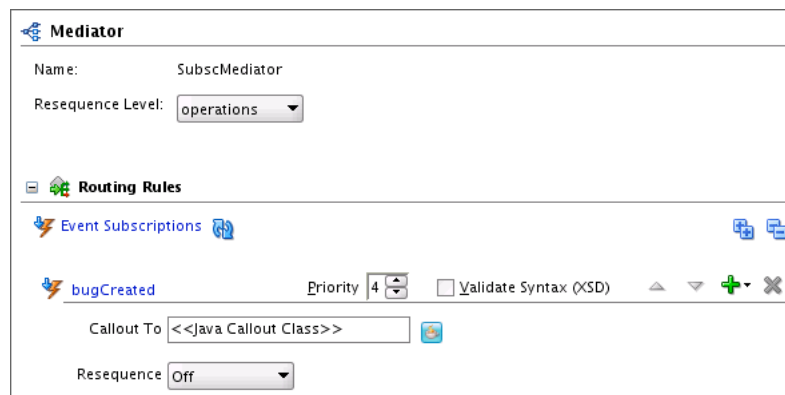
When you view the Oracle Mediator component in the SOA Composite Editor, the icon on the left side of the Oracle Mediator indicates that this Oracle Mediator is configured for an event subscription, as shown in [Figure 18-22](#).

Figure 18-22 Oracle Mediator Component Created with the Subscribe to Events Template



When you double-click the Oracle Mediator, the Mediator Editor appears, as shown in [Figure 18-23](#).

Figure 18-23 Event Subscription Oracle Mediator in the Mediator Editor



18.4.7 What You May Need to Know About the Mediator Editor

This section provides information you should know about creating an Oracle Mediator service component.

18.4.7.1 Resequencing

The resequencing feature of the Oracle Mediator reorders sets of messages that might arrive to the Oracle Mediator in the wrong sequence. You can define resequencing for all operations in an Oracle Mediator or for a specific operation. The resequencer provides three resequencing strategies that reorder incoming messages based on the type of sequencing information they contain.

For more information about resequencing in Oracle Mediator, see [Chapter 18, "Getting Started with Oracle Mediator."](#)

18.4.7.2 Routing Rules

Routing rules are mediation logic or execution logic that you define to achieve the requisite mediation. Below is an overview of the routing rule features; for more information about defining routing rules, see [Section 19.2, "Defining Routing Rules."](#)

You can specify the following to create a routing rule:

- Operation or Event

A routing rule can be triggered either by a service operation or an event subscription. The service operation can be synchronous, asynchronous, or one-way.

- Java Callouts

Java callouts perform external Java logic at various points in the execution of the Oracle Mediator.

- Static Routing Rule

A static routing rule is not expected to change depending on the invocation context. In this case, the routing can be an echo, a routing to another service, or a publishing of an event.

Static routing rules include the following information:

- Request Handler

This defines how Oracle Mediator handles incoming requests.

- Reply Handler

This defines how the synchronous response from the called service is handled by Oracle Mediator.

- Fault Handler

This defines how the named or declared faults from the called service are handled by Oracle Mediator.

- Callback Handler

This defines how the asynchronous response and callback from the called service are handled by Oracle Mediator.

- Timeout Handler in Callback

This defines how long Oracle Mediator waits for the asynchronous response and callback before performing timeout handling for the particular asynchronous request.

- Event Publishing and Service Invocation

This calls other services or publishes an event depending on the configuration of the handlers.

- Sequential or Parallel Execution

Each routing rule execution can be configured to be either sequential (that is, running in the same thread) or parallel (that is, running in different threads).

Note: For synchronous service invocations, the routing rule should always be sequential.

- Filter Expression

This defines a filter to be applied to the message before a rule is executed. Filters use XPath standards and enable selective execution of Oracle Mediator routing rules.

- Semantic Validation

This uses the Schematron validation standard to define semantic validation of incoming requests. Semantic validation also verifies that the data is correct.

- Transformation

This transforms incoming data to a format that is compliant with the called services or published events. Transformation is based on XSL transformation standards.

- Assign

This manipulates headers and properties for a message to meet the requirements of the called service.

- Dynamic Routing Rule

A dynamic routing rule lets you externalize the routing logic to an Oracle Rules Dictionary, which in turn enables dynamic modification of the routing logic in a routing rule. This feature depends on a decision service and Oracle Rules to obtain the routing logic at runtime.

Note: Oracle recommends using a Unicode database with AL32UTF8 as the database character set for full globalization support in Oracle Mediator.

18.5 Generating a WSDL File

You can generate the WSDL file for a message using an XML schema definition (XSD) file or using a sample file. When working with Oracle Mediator, you can generate a WSDL file at either of the following times:

- When you are creating an Oracle Mediator and you select the **Interface Definition from WSDL** template in the Create Mediator dialog, selecting **Generate WSDL from Schema(s)** next to the **WSDL URL** field opens the Create WSDL dialog.
- When you have an Oracle Mediator with no interface defined and you click **Define Service** next to the **WSDL URL** field in the Mediator Editor, selecting **Generate WSDL from Schema(s)** next to the **WSDL URL** field opens the Create WSDL dialog.

The Create WSDL dialog populates standard fields, such as the file name, directory, and namespace; and the dialog changes depending on the interface type you select. You can specify the same or different schema files for the message inputs.

18.5.1 How to Generate a WSDL File

The way you configure a WSDL file depends on the type of interface being defined by the WSDL file. You can define a one-way interface, a synchronous interface, or an asynchronous interface.

To generate a WSDL file for a one-way interface from an XSD file:

Perform these steps after the Create WSDL dialog appears when you are creating an Oracle Mediator component or when you are defining a service for an Oracle Mediator component.

- On the Create WSDL dialog, accept the default values or enter the following information for the WSDL file:
 - File Name:** A unique name for the WSDL file.
 - Directory:** The directory where you want to store the WSDL file. By default, it is stored in the same location as the Oracle Mediator file. This must be the current project directory or one of its subdirectories. If the specified directory does not exist, Oracle JDeveloper creates it.
 - Namespace:** A namespace address for the WSDL file; for example, `http://oracle.com/esb/namespaces/Mediator`.
The namespace that you specify is defined as the `tns` namespace in the WSDL file.
 - Port Type:** The name of the port type in the WSDL file that contains the operation to use.
 - Operation:** The name of the action to perform; for example, `executeQuery`.

Note: Spaces and special characters are not allowed in an operation name or port type. Only alphabetic and numeric characters are supported, and the first character cannot be a number.

- In the **Interface Type** field, select **One-Way Interface**.

The **Input** field appears, as shown in [Figure 18–24](#).

Figure 18–24 Create WSDL Dialog for a One-Way Interface

The screenshot shows the 'Create WSDL' dialog box with the following values:

- File Name:** ReadFile.wsdl
- Directory:** oraMiddlewarePS3/jdeveloper/jdev/bin/v/mywork/TimeWindow/Project1
- Namespace:** http://xmlns.oracle.com/TimeWindow/Project1/ReadFile
- Port Type:** execute_ptt
- Operation:** execute
- Interface Type:** One-Way Interface

The **Input** section contains a table with the following header:

Message Part Name	Element or Type	Schema URL

At the bottom, the checkbox **Generate partnerlinkType extension** is checked. Buttons for **Help**, **OK**, and **Cancel** are at the bottom right.

- To the upper right of the **Input** field, click **Add a new message part**.

The Add Message Part dialog appears, as shown in [Figure 18-25](#).

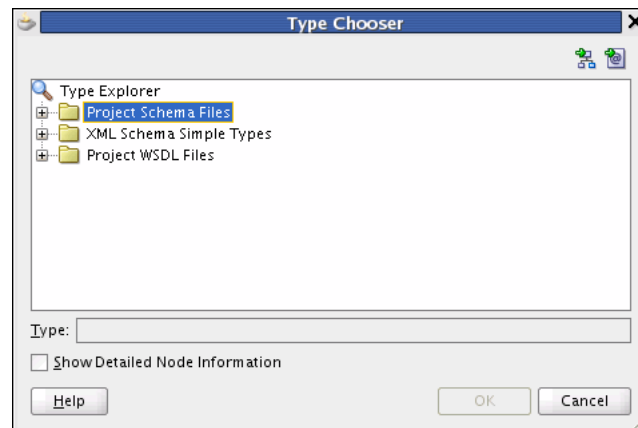
Figure 18-25 Add Message Part Dialog



4. In the **Part Name** field, enter a name for the message part.
5. To the right of the **URL** field, click the **browse for schema file** icon to browse for the URL.

The Type Chooser dialog appears and contains a list of the schema files (XSD files), as shown in [Figure 18-26](#).

Figure 18-26 Type Chooser Dialog



6. Expand the Type Explorer tree to locate and select the schema element to use.
If the schema you want to use is not located in the project in which you are working, you can import a schema XSD file or WSDL file into the project using the **Import Schema File** or **Import WSDL** icon in the upper right corner of the dialog.

Note: If you want to use a schema XSD file that resides on your local file system, ensure that the XSD file and any XSD files that it imports all reside in the Oracle JDeveloper project directory. This ensures that the schema is deployed with the project and is made available at runtime.

After you specify a file, Oracle JDeveloper parses it to determine the defined schema elements and displays them in a list from which you select.

7. Select the root element of the XSD file and click **OK**.

The Add Message Part dialog reappears with the **URL** and **Schema Element** fields populated from the Type Chooser dialog. If you selected an XSD simple type, these fields are replaced by a **Simple Type** field.

8. Click **OK** on the Add Message Part dialog.

The input information appears in the **Input** field of the Create WSDL dialog.

9. If needed, repeat the above steps to define additional message parts.
10. Click **OK**.

Note: Partner link types are generally used in BPEL, so you do not need to select **Generate partnerlinkType extension** for Oracle Mediator.

To generate a WSDL file for a synchronous interface from an XSD file:

Perform these steps after the Create WSDL dialog appears when you are creating an Oracle Mediator component or when you are defining a service for an Oracle Mediator component.

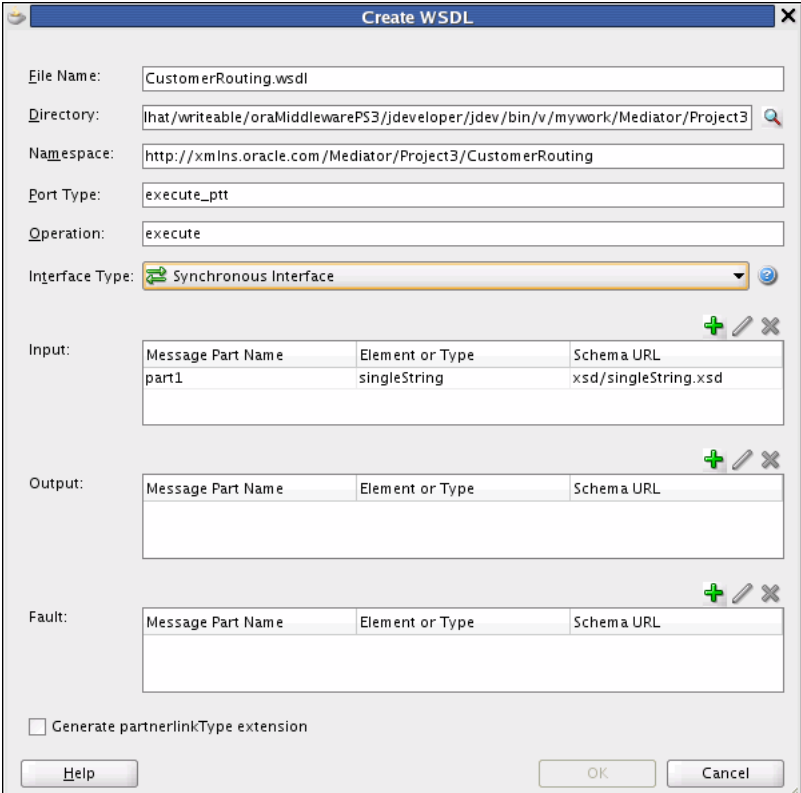
1. On the Create WSDL dialog, enter the following information for the WSDL file:
 - **File Name:** A unique name for the WSDL file.
 - **Directory:** The directory where you want to store the WSDL file. By default, it is stored in the same location as the Oracle Mediator file.
 - **Namespace:** A namespace address for the WSDL file; for example, `http://oracle.com/esb/namespaces/Mediator`.

The namespace that you specify is defined as the `tns` namespace in the WSDL file.
 - **Port Type:** The name of the port type in the WSDL file that contains the operation to use.
 - **Operation:** The name of the action to perform; for example, `executeQuery`.

Note: Spaces and special characters are not allowed in an operation name or port type. Only alphabetic and numeric characters are supported, and the first character cannot be a number.

2. In the **Interface Type** field, select **Synchronous Interface**.

The **Input**, **Output**, and **Fault** fields appear, as shown in [Figure 18-27](#).

Figure 18-27 Create WSDL Dialog for a Synchronous Interface


The **Create WSDL** dialog box is shown with the following fields and values:

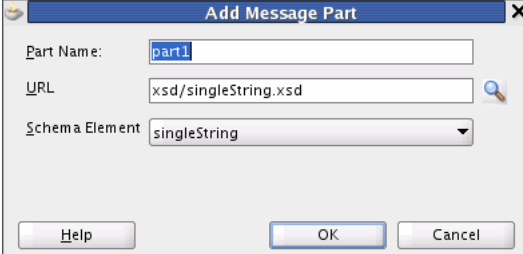
- File Name:** CustomerRouting.wsdl
- Directory:** /hat/writeable/oraMiddlewarePS3/jdeveloper/jdev/bin/v/mywork/Mediator/Project3
- Namespace:** http://xmlns.oracle.com/Mediator/Project3/CustomerRouting
- Port Type:** execute_ptt
- Operation:** execute
- Interface Type:** Synchronous Interface

Below these fields are three tables for **Input**, **Output**, and **Fault** message parts. Each table has columns for Message Part Name, Element or Type, and Schema URL.

Message Part Name	Element or Type	Schema URL
part1	singleString	xsd/singleString.xsd

The **Output** and **Fault** tables are currently empty. At the bottom, there is a checkbox for **Generate partnerlinkType extension** (unchecked) and buttons for **Help**, **OK**, and **Cancel**.

- To the upper right of the **Input** field, click **Add a new message part**.
The Add Message Part dialog appears, as shown in [Figure 18-28](#).

Figure 18-28 Add Message Part Dialog


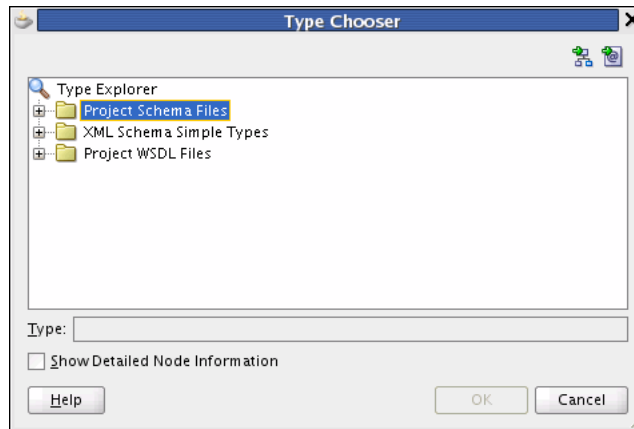
The **Add Message Part** dialog box is shown with the following fields and values:

- Part Name:** part1
- URL:** xsd/singleString.xsd
- Schema Element:** singleString

Buttons for **Help**, **OK**, and **Cancel** are at the bottom.

- In the **Part Name** field, enter a name for the message part.
- To the right of the **URL** field, click the **browse for schema file** icon to browse for the URL.

The Type Chooser dialog appears and contains a list of the schema files (XSD files), as shown in [Figure 18-29](#).

Figure 18–29 Type Chooser Dialog

6. Expand the Type Explorer tree to locate and select the schema element to use.

If the schema you want to use is not located in the project in which you are working, you can import a schema XSD file or WSDL file into the project using the **Import Schema File** or **Import WSDL** icon in the upper right corner of the dialog.

Note: If you want to use a schema XSD file that resides on your local file system, ensure that the XSD file and any XSD files that it imports all reside in the Oracle JDeveloper project directory. This ensures that the schema is deployed with the project and is made available at runtime.

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.

7. Select the root element of the XSD file and click **OK**.

The Add Message Part dialog reappears with the **URL** and **Schema Element** fields populated from the Type Chooser dialog. If you selected an XSD simple type, these fields are replaced by a **Simple Type** element.

8. Click **OK** on the Add Message Part dialog.

The input information appears in the **Input** field of the Create WSDL dialog.

9. Repeat the above steps to define message parts for the **Output** and **Fault** fields.

The output represents the response message and is required in synchronous transactions. **Faults** are optional.

10. Click **OK**.

Note: Partner link types are generally used in BPEL, so you do not need to select **Generate partnerlinkType extension** for Oracle Mediator.

To generate a WSDL file for an asynchronous interface from an XSD file:

Perform these steps after the Create WSDL dialog appears when you are creating an Oracle Mediator component or when you are defining a service for an Oracle Mediator component.

- On the Create WSDL dialog, enter the following information for the WSDL file:
 - File Name:** A unique name for the WSDL file.
 - Directory:** The directory where you want to store the WSDL file. By default, it is stored in the same location as the Oracle Mediator file.
 - Namespace:** A namespace address for the WSDL file; for example, `http://oracle.com/esb/namespaces/Mediator`.
The namespace that you specify is defined as the `tns` namespace in the WSDL file.
 - Port Type:** The name of the port type in the WSDL file that contains the operation to use.
 - Operation:** The name of the action to perform; for example, `executeQuery`.

Note: Spaces and special characters are not allowed in an operation name or port type. Only alphabetic and numeric characters are supported, and the first character cannot be a number.

- In the **Interface Type** field, select **Asynchronous Interface**.

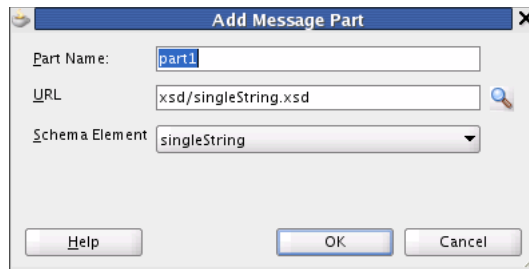
The **Input** field and **Callback** section appear, as shown in [Figure 18–30](#).

Figure 18–30 Create WSDL Dialog for an Asynchronous Interface

The screenshot shows the 'Create WSDL' dialog box with the following details:

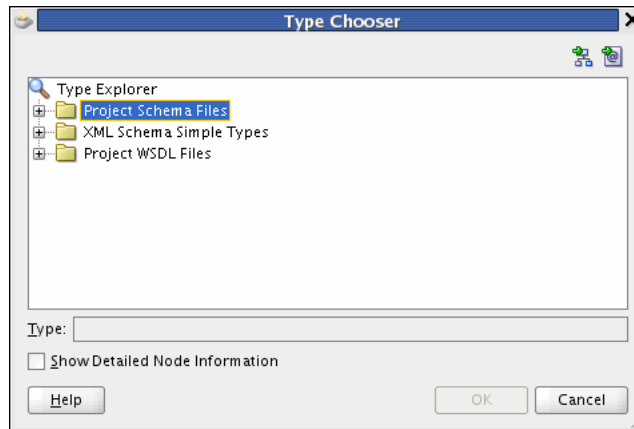
- File Name:** CustomerRouting.wsdl
- Directory:** lhat/writeable/oraMiddlewarePS3/jdeveloper/jdev/bin/v/mywork/Mediator/Project3
- Namespace:** http://xmlns.oracle.com/Mediator/Project3/CustomerRouting
- Port Type:** execute_ptt
- Operation:** execute
- Interface Type:** Asynchronous Interface
- Input Section:** Contains a table with columns: Message Part Name, Element or Type, Schema URL.
- Callback Section:**
 - Port Type:** callback_ptt
 - Operation:** callback
- Bottom:** A checkbox for 'Generate partnerlinkType extension' and buttons for 'Help', 'OK', and 'Cancel'.

- To the upper right of the first **Input** field, click **Add a new message part**.
The Add Message Part dialog appears, as shown in [Figure 18–31](#).

Figure 18–31 Add Message Part Dialog

4. In the **Part Name** field, enter a name for the message part.
5. To the right of the **URL** field, click the **browse for schema file** icon to browse for the URL.

The Type Chooser dialog appears and contains a list of the schema files (XSD files), as shown in [Figure 18–32](#).

Figure 18–32 Type Chooser Dialog

6. Expand the Type Explorer tree to locate and select the schema element to use.

If the schema you want to use is not located in the project in which you are working, you can import a schema XSD file or WSDL file into the project using the **Import Schema File** or **Import WSDL** icon in the upper right corner of the dialog.

Note: If you want to use a schema XSD file that resides on your local file system, ensure that the XSD file and any XSD files that it imports all reside in the Oracle JDeveloper project directory. This ensures that the schema is deployed with the project and is made available at runtime.

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.

7. Select the root element of the XSD file and click **OK**.

The Add Message Part dialog reappears with the **URL** and **Schema Element** fields populated from the Type Chooser dialog. If you selected an XSD simple type, these fields are replaced by a **Simple Type** element.

8. Click **OK** on the Add Message Part dialog.

The input information appears in the **Input** field of the Create WSDL dialog.

9. Repeat the above steps to define the input message parts for the Callback section.

Note: The callback input represents the response message and is required in asynchronous transactions.

10. In the Callback section, specify the following information for the response message:

- **Port Type:** The name of the port type in the WSDL file that contains the operation to use.
- **Operation:** The name of the action to perform; for example, `executeResponse`.

Note: Spaces and special characters are not allowed in an operation name or port type. Only alphabetic and numeric characters are supported, and the first character cannot be a number. Both of these fields are required.

11. Click OK.

Note: Partner link types are generally used in BPEL, so you do not need to select **Generate partnerlinkType extension** for Oracle Mediator.

To generate the WSDL file based on a sample file:

You can generate a WSDL file from a file in a native format such as a comma-separated value (CSV) file, a fixed-length file, a document type definition (DTD) file, or a COBOL copybook file. Use the Native Format Builder wizard to generate a WSDL file based on a sample file. The Native Format Builder wizard appears when you click **Define Schema for Native Format** in the **Request**, **Response**, **Fault**, and **Callback** tabs of the Create WSDL dialog. A WSDL file is generated after you complete the wizard.

For information about the Native Format Builder wizard, see the *Oracle Fusion Middleware User's Guide for Technology Adapters*.

18.6 Specifying Operation or Event Subscription Properties

After creating an Oracle Mediator, you can use the Mediator Editor to select the **Validate Syntax (XSD)** checkbox for an operation or event subscription. You can select this option to validate the schemas of the inbound messages. By default, this checkbox is not selected.

18.7 Modifying an Oracle Mediator Service Component

You can modify the operations or event subscriptions of an Oracle Mediator using the Mediator Editor.

18.7.1 How To Modify Operations of an Oracle Mediator

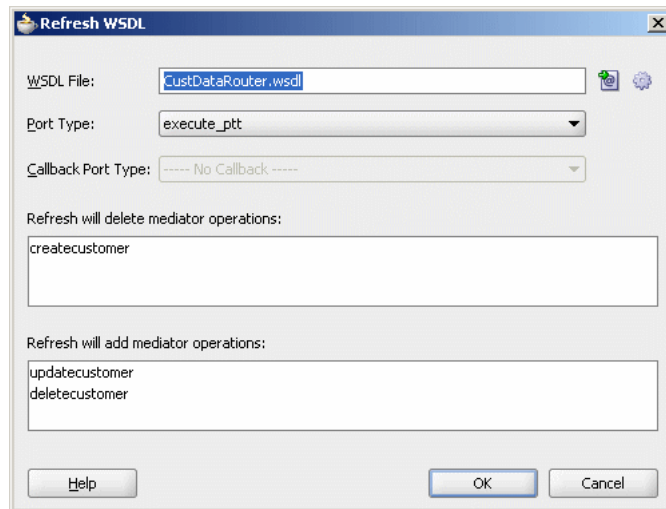
You can modify an Oracle Mediator WSDL file by adding or deleting operations. After modifying the WSDL file, use the Refresh WSDL dialog to synchronize the changes.

To modify the operations of an Oracle Mediator:

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

The Refresh WSDL dialog appears. If you have made any modifications to the WSDL file, the Refresh WSDL dialog lists all the operations to delete or add. The **Refresh will delete Mediator operation** field lists all the operations that have been removed from the WSDL file. The **Refresh will add Mediator operation** field lists all the new operations that have been added in the WSDL file. [Figure 18–33](#) shows the Refresh WSDL dialog.

Figure 18–33 Refresh WSDL Dialog



2. To specify a different WSDL file, click **Find existing WSDLs** to the right of the **WSDL URL** field 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.

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

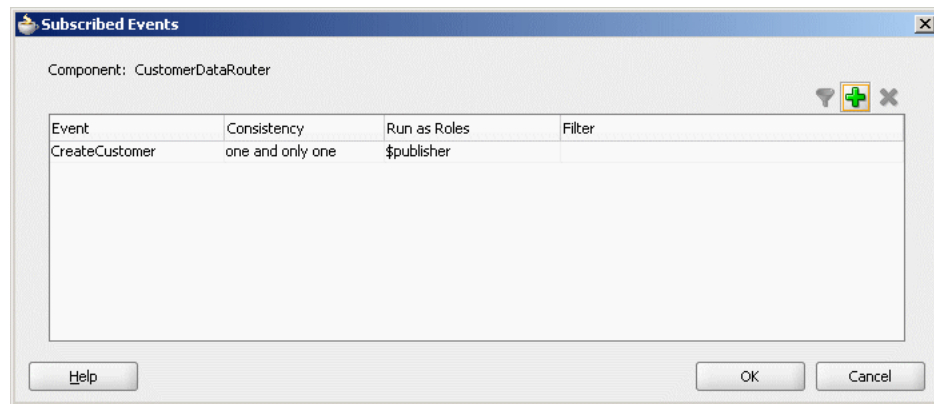
18.7.2 How To Modify Event Subscriptions of an Oracle Mediator

You can subscribe to new events, modify existing event subscriptions, and unsubscribe from subscribed events using the **Manage Event Subscriptions** option in the Mediator Editor.

To modify event subscriptions of an Oracle Mediator:

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

The Subscribed Events dialog appears, as shown in [Figure 18–34](#).

Figure 18–34 The Subscribed Events Dialog

2. You can perform any of the following functions:

- Subscribe to a new event.
- Unsubscribe from an event.
- Modify or specify the filter criteria for an event.
- Modify the **Consistency** or **Run as Roles** properties of an event subscription.

For more information about the **Consistency**, **Run as Roles**, and **Filter** fields of an event, see [Section 18.4.6, "Creating an Oracle Mediator for an Event Subscription."](#)

3. Click **OK**.

4. From the **File** menu, select **Save All**.

Creating Oracle Mediator Routing Rules

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

This chapter includes the following sections:

- [Section 19.1, "Introduction to Routing Rules"](#)
- [Section 19.2, "Defining Routing Rules"](#)
- [Section 19.3, "Creating an Oracle Mediator for Routing Messages"](#)
- [Section 19.4, "Creating an Asynchronous Request and Response Using Oracle Mediator"](#)

19.1 Introduction to Routing Rules

Oracle Mediator lets you route data between service consumers and service providers. As the data flows from service to service, it must be transformed. These two tasks, routing and transformation, are the core responsibilities of Oracle Mediator. You can use routing rules to specify how a message processed by an Oracle Mediator reaches its next destination. Routing rules specify where an Oracle Mediator sends the message, how it sends the message, and what changes should be made to the message structure before sending it to the target service.

Routing rules can be of the following two types:

- **Static Routing Rules**
Static rules do not change depending on the invocation context and are applied consistently.
- **Dynamic Routing Rules**
Dynamic rules let you externalize the routing logic to an Oracle Rules Dictionary, which in turn enables dynamic modification of the routing logic.

For more information about creating routing rules, see [Section 19.2.2, "How to Create Static Routing Rules"](#) and [Section 19.2.3, "How to Create Dynamic Routing Rules."](#)

19.2 Defining Routing Rules

Routing rules can only be defined for an Oracle Mediator with a defined interface. For more information on how to define an interface, see [Section 18.4.1.3, "How to Define an Interface for an Oracle Mediator."](#)

19.2.1 How To Access the Routing Rules Section

You define the routing rules in the **Routing Rules** section of the Mediator Editor.

To access the routing rules section:

You can access the **Routing Rules** section of the Mediator Editor using one of the following methods:

- From the SOA Composite Editor:
 - a. Double-click the icon that represents the Oracle Mediator for which you want to specify the routing rules.
 - b. If the **Routing Rules** section is not visible, click the **Plus (+)** icon next to **Routing Rules**.
- From the Application Navigator:
 - a. In the Application Navigator, expand the SOA project and then expand the **SOA Content** folder.
 - b. In the **SOA Content** folder, double-click the name of the Oracle Mediator file in which you want to specify the routing rules.
The Oracle Mediator file has an **MPLAN** extension.
 - c. If the **Routing Rules** section is not visible, click the **Plus (+)** icon next to **Routing Rules**.

Figure 19–1 shows the **Routing Rules** section of the Mediator Editor.

Figure 19–1 Mediator Editor- Routing Rules Section

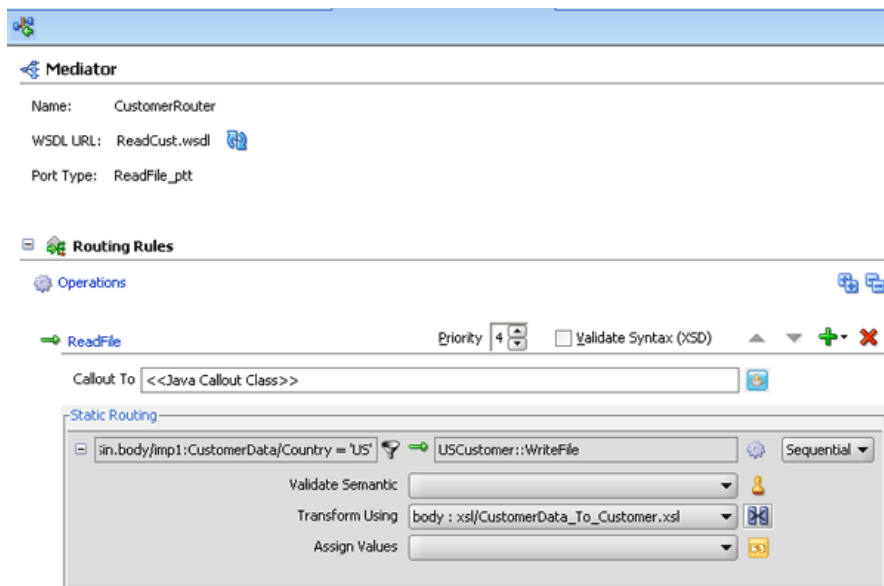
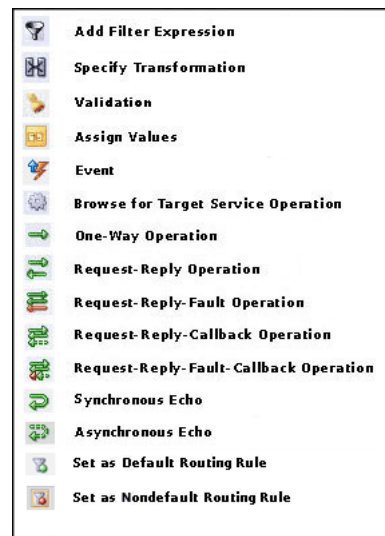


Figure 19–2 lists and describes the icons in the **Routing Rules** section.

Figure 19–2 Routing Rule Section Icons



19.2.2 How to Create Static Routing Rules

When you define static routing rules, you can configure the following information and types of rules:

- Target service

Oracle Mediator sends messages to the target service you specify. This service can either be defined as a WSDL interface or a Java interface. For information about invoking a target service, see [Section 19.2.2.1, "How to Specify Oracle Mediator Services or Events"](#).

- Execution type

Oracle Mediator executes routing rules either sequentially or in parallel. For information about specifying an execution type, see [Section 19.2.2.3, "How to Specify Sequential or Parallel Execution"](#).

- Reply, callback, and fault handlers

You can define how Oracle Mediator handles synchronous reply, callback, and fault messages. For information about handlers, see [Section 19.2.2.4, "How to Configure Response Messages"](#) and [Section 19.2.2.6, "How to Handle Faults"](#).

- Filter expression

You can define a filter expression that is applied to the message content (payload or headers). When you define a filter, the contents are analyzed before any service is invoked. For example, you might apply a filter expression that specifies that a service be invoked only if the message includes a customer ID, or if the value for that customer ID matches a certain pattern. For information about specifying filter expressions, see [Section 19.2.2.7, "How to Specify an Expression for Filtering Messages"](#).

- Transformations

Oracle Mediator can transform message data before forwarding the message to a service. You can define transformations to set a value on the target payload by mapping data or by assigning values.

The XSLT Mapper lets you to define transformations that apply to the whole message body to convert messages from one XML schema to another. The Assign Values function works on individual fields. Using this dialog, you can assign values from the message (for example, payload and headers), from a constant, or from various system properties, such as the properties of an adapter present in the data path. For information about defining transformations, see [Section 19.2.2.8, "How to Create Transformations"](#) and [Section 19.2.2.9, "How to Assign Values"](#).

- Accessing header variables from expressions

Oracle Mediator can detect any SOAP headers that are used in building the expression for the current routing rule operation. For information about accessing headers, see [Section 19.2.2.11, "How to Access Headers for Filters and Assignments"](#) and [Section 19.2.2.11.2, "Manual Expression Building for Accessing Properties for Filters and Assignments"](#).

- Schematron-based validations

You can specify the Schematron files that Oracle Mediator should use to validate different parts of an inbound message. For information about performing Schematron-based validations, see [Section 19.2.2.12, "How to Use Semantic Validation"](#).

- Java callout

Custom Java class callouts let you use regular expressions with Java code, when regular expressions alone do not suffice. For information about using Java callouts, see [Section 19.2.2.13, "How to Use Java Callouts"](#).

- User-defined extension functions

These are your own set of functions that can be used by the XSLT Mapper. For information about using user-defined extension functions, see ["To add user-defined extension functions:"](#).

19.2.2.1 How to Specify Oracle Mediator Services or Events

After creating an Oracle Mediator component, you associate it with inbound service operations or event subscriptions and with outbound targets. Targets are outbound service operations or event publishing. A target specifies the next service or event to which an Oracle Mediator sends messages and also specifies which service operation to invoke. You can specify a service or an event as a target type.

You can also echo source messages back to the initial caller after any transformation, validations, assignments, or sequencing operations are performed. An echo can only be specified if the Oracle Mediator component has a synchronous or asynchronous interface. Whether the echo is synchronous or asynchronous depends on the WSDL file of the caller. The echo option is only available for inbound service operations and is not available for event subscriptions.

The purpose of the echo option is to expose all the Oracle Mediator functionality as a callable service without having to route it to any other service. For example, you can call an Oracle Mediator to perform a transformation, a validation, or an assignment, and then echo the Oracle Mediator back to your application without routing it anywhere else.

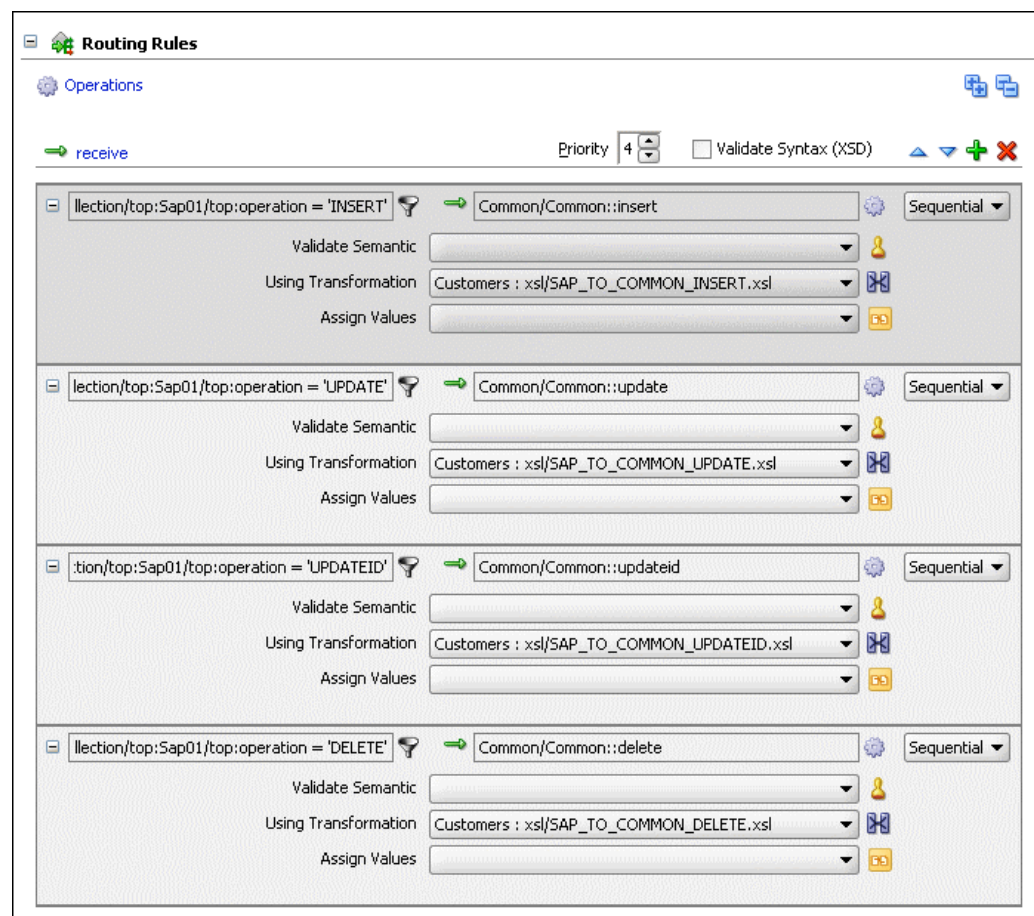
You can specify multiple routings for an inbound operation or event. Each routing is mapped to one target service invocation or event. Therefore, to specify multiple

service invocations or raise multiple events, you must specify one routing rule for each target. For example, you can invoke an operation based on a message payload from the following operations defined in a service:

- insert
- update
- updateid
- delete

To do this action, you must create four routing rules, one for each operation. Later, when you specify a filter expression for each rule, you can specify which target and operation is applied to each message instance based on the message payload, as shown in [Figure 19–3](#).

Figure 19–3 Multiple Routings for an Inbound Operation



To invoke a service:

To perform this step, the target service must be defined in a WSDL document or a Java interface.

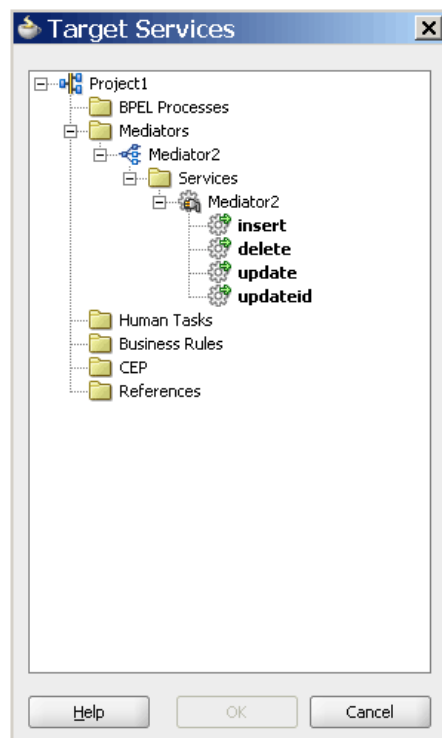
1. In the **Routing Rules** section, click **Add** next to the operation for which you are defining routing rules, and then select **static routing rule**.

The Target Type dialog appears, as shown in [Figure 19–4](#).

Figure 19–4 Target Type Dialog

2. Click **Service**.

The Target Services dialog appears, as shown in [Figure 19–5](#).

Figure 19–5 Target Services Dialog

3. In the Target Services dialog, navigate to and then select an operation provided by a service.

Note: You can select a service defined by a WSDL file or a Java interface. A service can consist of multiple operations, as shown in [Figure 19–5](#).

4. Click **OK**.
5. If you selected a target service defined by a Java interface, the Interface Required dialog appears. Click **Yes** to create the required WSDL file, and then click **OK** on the confirmation dialog.

A new Static Routing section appears where you can define the routing rule.

6. Configure the routing rule as described the remaining sections of this chapter.

To trigger an event:

1. In the **Routing Rules** section, click **Add** next to the operation for which you are defining routing rules, and then select **static routing rule**.

The Target Type dialog appears, as shown in [Figure 19–4](#).

2. Click **Event**.

The Event Chooser dialog appears.

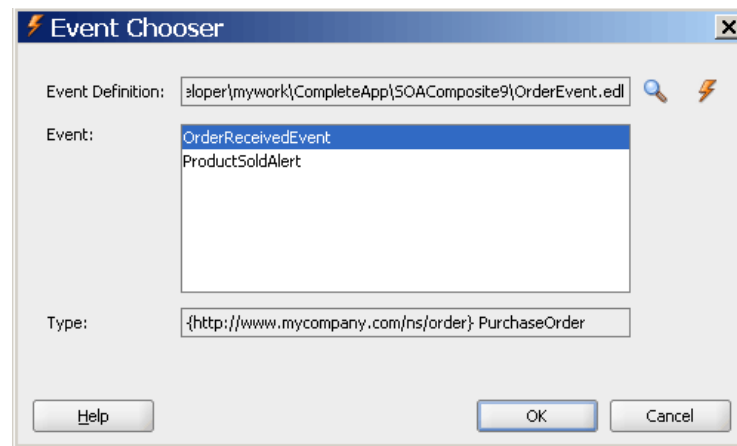
3. To the right of the **Event Definition** field, click **Search**.

The SOA Resource Browser dialog appears.

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

The **Event** field is populated with the events defined in the selected file, as shown in [Figure 19–6](#).

Figure 19–6 Event Chooser Dialog



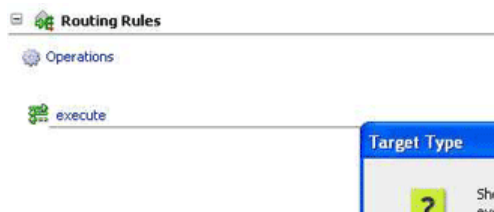
Note: Instead of browsing for an existing event definition file, you can create a new file by clicking **Create new event definition (edl) file** and completing the fields in the Create Event Definition File dialog.

5. Select an event.
6. Click **OK**.
A new Static Routing section appears where you can define the routing rule.
7. Configure the routing rule as described the remaining sections of this chapter.

To echo a service:

1. In the **Routing Rules** section, click **Add** next to the operation for which you are defining routing rules, and then select **static routing rule**.

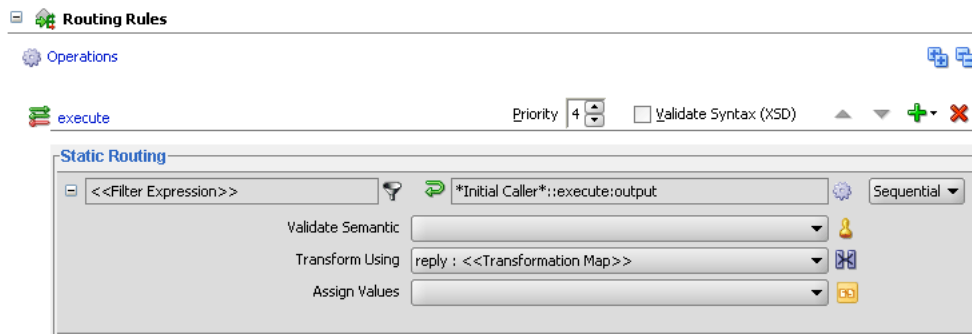
The Target Type dialog is displayed, as shown in [Figure 19–7](#).

Figure 19–7 Target Type Dialog

2. Click Echo.

Note: The Echo button only appears on the Target Type dialog if the interface is synchronous or asynchronous.

Figure 19–8 shows a routing rule with a synchronous echo. An asynchronous echo has an icon with a dotted line on the return.

Figure 19–8 Sample Oracle Mediator Supporting Echo Operation

19.2.2.2 What You May Need to Know About Echoing a Service

The echo option has the following limitations:

- Echoing a service is supported only with Oracle Mediator interfaces having the following types of WSDL files:
 - Request/reply
 - Request/reply/fault
 - Request/callback

Note: The echo option is not available for Oracle Mediator interfaces having request/reply/fault/callback WSDL files or for one-way WSDL files.

- The echo option is available for synchronous operations like request/reply and request/reply/fault.

Note: The echo option is only available for synchronous operations when the routing rule is sequential because parallel routing rules are not supported for Oracle Mediators with synchronous operations.

- For synchronous operations with a conditional filter, the echo option does not return a response to the caller when the filter condition is set to `false`. Instead, it returns a `null` response.
- The echo option is available for asynchronous operations *only if* the Oracle Mediator interface has a callback operation. In this case, the echo is run on a separate thread.

Note: The asynchronous echo option is available only when the routing rule is parallel. If you use the echo option, then sequential routing rules are not supported for Oracle Mediators with asynchronous operations.

19.2.2.3 How to Specify Sequential or Parallel Execution

A routing rule can be executed either in parallel or sequentially. To specify an execution type for a routing rule, select the **Sequential** or **Parallel** execution type in the **Routing Rules** section.

Basic Principles of Sequential Routing Rules

Oracle Mediator processes sequential routing rules based on the following principles:

- Oracle Mediator evaluates routings and performs the resulting actions sequentially. Sequential routings are evaluated in the same thread and transaction as the caller.
- Oracle Mediator always enlists itself into the global transaction propagated through the thread that is processing the incoming message. For example, if an inbound JCA adapter invokes an Oracle Mediator, the Oracle Mediator enlists itself with the transaction that the JCA adapter has initiated.
- Oracle Mediator propagates the transaction through the same thread as the target components while executing the sequential routing rules.
- Oracle Mediator never commits or rolls back transactions propagated by external entities.
- Oracle Mediator manages the transaction only if the thread-invoking Oracle Mediator does not already have an active transaction. For example, if Oracle Mediator is invoked from inbound SOAP services, Oracle Mediator starts a transaction and commits or rolls back the transaction depending on success and failure.

Basic Principles of Parallel Routing Rules

Oracle Mediator processes routing rules in parallel based on the following principles:

- Oracle Mediator queues and evaluates routings in parallel in different threads.
The messages of each Oracle Mediator service component are retrieved in a weighted, round-robin fashion to ensure that all Oracle Mediator service components receive parallel processing cycles. This is true even if one or more Oracle Mediator service components produce a higher number of messages compared to other components. The weight used is the message priority set when designing an Oracle Mediator service component. Higher numbers of parallel processing cycles are allocated to the components that have higher message priority.

You can set the **Priority** field in the Mediator Editor to indicate the priority of an Oracle Mediator service component. Priorities can range from zero to nine, with nine being the highest priority. The default priority is four.

Note: The **Priority** property is applicable only to parallel routing rules.

- Oracle Mediator initiates a new transaction for processing each parallel rule. The initiated transaction ends with an enqueue to the Oracle Mediator parallel message dehydration store.

For example, if an Oracle Mediator service component has one parallel routing rule, one message is enqueued on the Oracle Mediator parallel message dehydration store. The parallel message dispatcher to the store then initiates a transaction, reads the message from the database store, and invokes the target component or service of this routing rule. The transaction initiated by the listener thread is a completely new transaction and is propagated to the target components.

Note: Dehydrating of messages means storing the incoming messages in a database for parallel routing rules so they can be processed later by worker threads.

- Oracle Mediator commits or rolls back transactions because it is the initiator of these transactions.

If an operation or event has both sequential and parallel routing rules, first sequential routing rules are evaluated and actions are performed, and then parallel routings are queued for parallel execution.

Note: If an Oracle Mediator service component with a request-response interface has only parallel routing rules, the Oracle Mediator service component does not send a response back to the caller. Though you can create this type of Oracle Mediator service component, the caller of the Oracle Mediator service component does not receive a response at runtime.

19.2.2.4 How to Configure Response Messages

In the Oracle Mediator routing rules, you can specify how to handle the response messages in synchronous and asynchronous interactions. For 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, or you can send them back to the initial caller, if the initial caller is expecting responses and faults.

For asynchronous interactions, you can specify transformations and assignments, and a timeout period for receiving the response. The timeout period can be specified in seconds, hours, days, months, or years. By default, the timeout period is infinite. If a callback response does not come within the specified timeout period, a timeout response can be forwarded to another service, to another event, or back to the initial caller.

You cannot route an Oracle Mediator response to a two-way service. If you want to route a response to a two-way service, you should use a one-way Oracle Mediator between the first Oracle Mediator and the two-way service. The response should first be forwarded to the one-way Oracle Mediator, which in turn should call the two-way service.

Notes:

- Zero is an unsupported value to be specified as a timeout period.
 - If the callback is received and processing of the callback fails, by default the timeout handler is invoked for processing the action specified in the timeout handler.
 - Typically, the caller receives the callback after waiting for 100 milliseconds. However, if you have a bridge Oracle Mediator with a sequential routing rule and a connection to a synchronous interface service, then due to the complex flow of the program with all sequential routing rules, the caller may take longer to get ready to receive the callback. You can work around this issue by changing the routing rule of the bridge Oracle Mediator to parallel.
-

To specify a timeout period for asynchronous processing:

The following steps are performed in the Routing Rules section of the Mediator Editor.

1. Next to the <<**Target Operation**>> field by the **Timeout in** field in the **Callback** section, click the **Browse for target service operation** icon.

The Target Type dialog appears.

2. Select **Service, Event, or Initial Caller**.

If you selected Service or Event, the Target Service or the Event Chooser appears depending on your selection.

3. Select an event or service.
4. Click **OK**
5. In the **Timeout in** field, enter the number of units for the timeout period, and then select the unit of time from the dropdown list.

The timeout response is forwarded to the specified service or event.

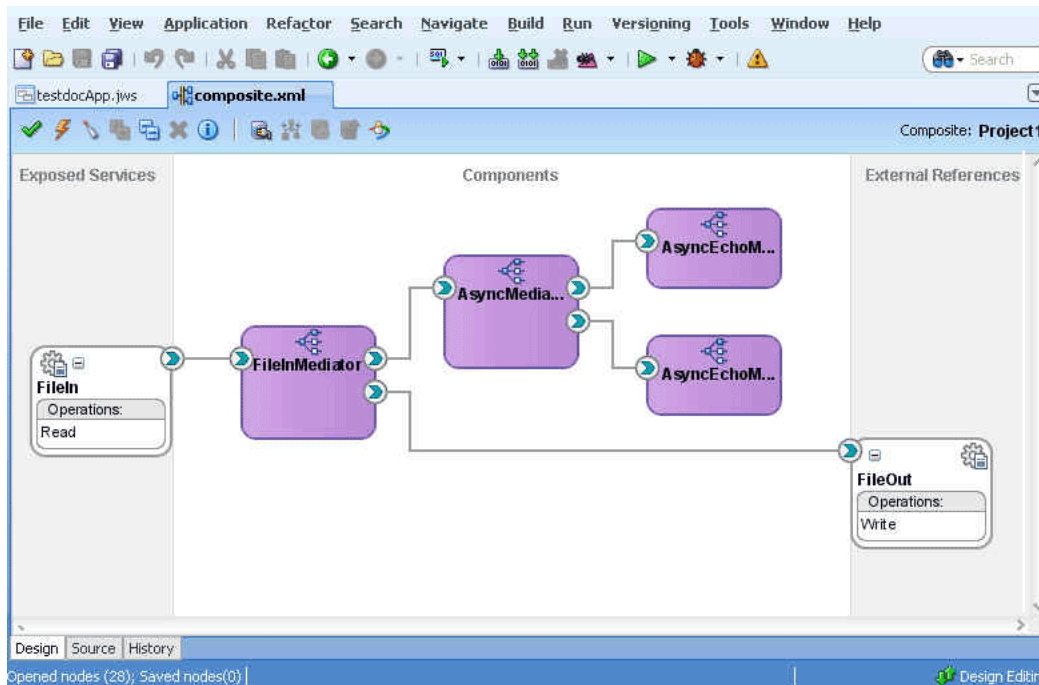
Note: If the number of routing rules is larger and the time taken to execute the routing rules exceeds the transaction timeout, you must set the transaction timeout to a value that is greater than the time taken to execute all the routing rules.

19.2.2.5 How to Handle Multiple Callbacks

A single Oracle Mediator cannot handle multiple callbacks. If you have a composite application with an Oracle Mediator that receives multiple callbacks, the behavior of the composite application is undetermined. For example, in the scenario shown in [Figure 19-9](#), **AsyncMediator** forwards the callback response from **AsyncEchoMediator1** and **AsyncEchoMediator2** to **FileInMediator**. In such a flow, the **AsyncMediator** might return the callback from both **AsyncEchoMediator1** and

`AsyncEchoMediator2`, or from either one of them. The exact behavior is random and unpredictable.

Figure 19–9 Sample Oracle Mediator Handling Multiple Callback



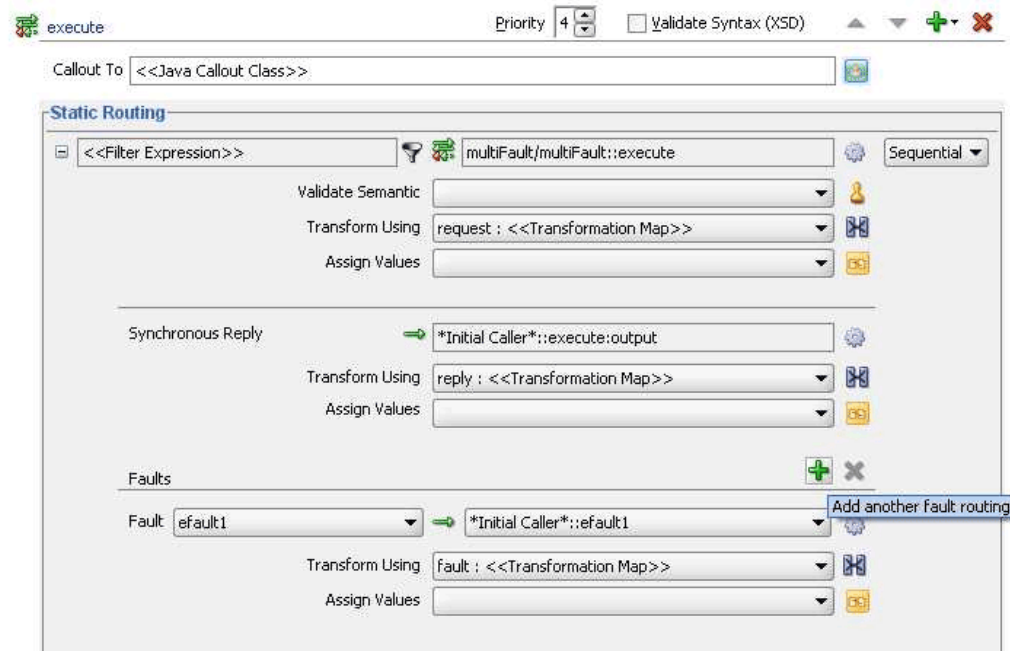
19.2.2.6 How to Handle Faults

If you create a new routing rule in which the target service operation has one or more faults, you still see a single fault routing section in the Mediator Editor. If the source Oracle Mediator service component supports one or more faults, then the fault is routed back to the caller by default. You can choose the source and target fault names to be routed. You can also use the service browser to route the fault to another target.

To define an additional fault routing:

The following steps are performed in the Routing Rules section of the Mediator Editor.

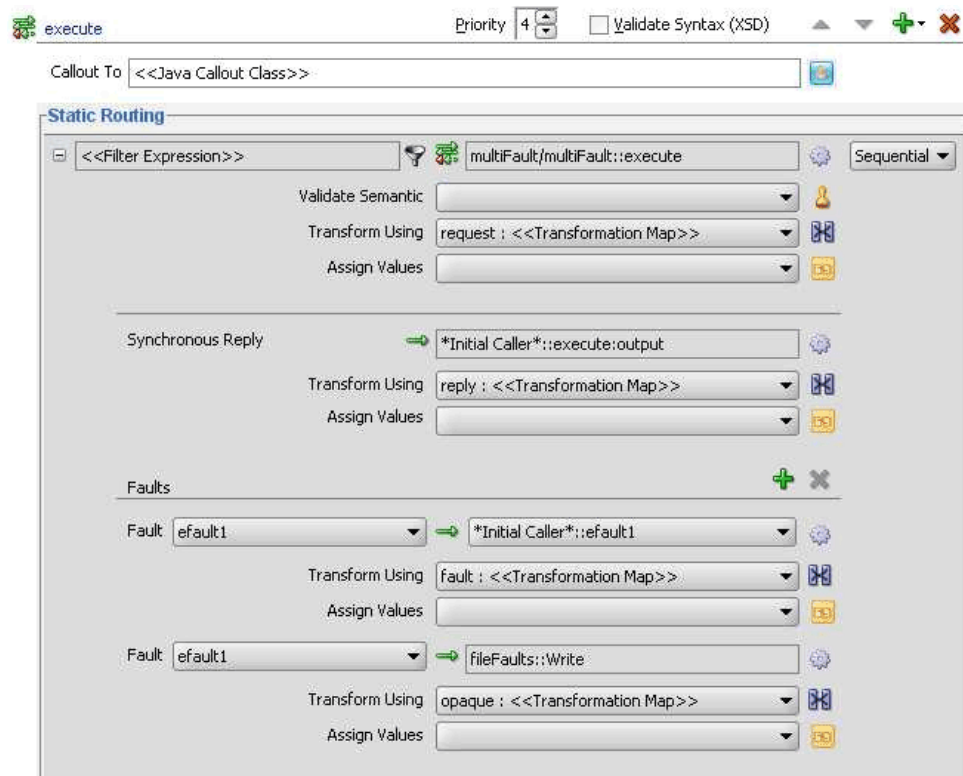
1. In the **Faults** section, click the **Add another fault routing** button shown in [Figure 19–10](#).

Figure 19–10 Adding a Second Fault

Another fault section appears in the routing rule box.

2. Configure the target service, transformations, and assign values for the new fault.

Figure 19–11 shows a second fault being routed to a file adapter service.

Figure 19–11 Second Fault Added to Routing Rules

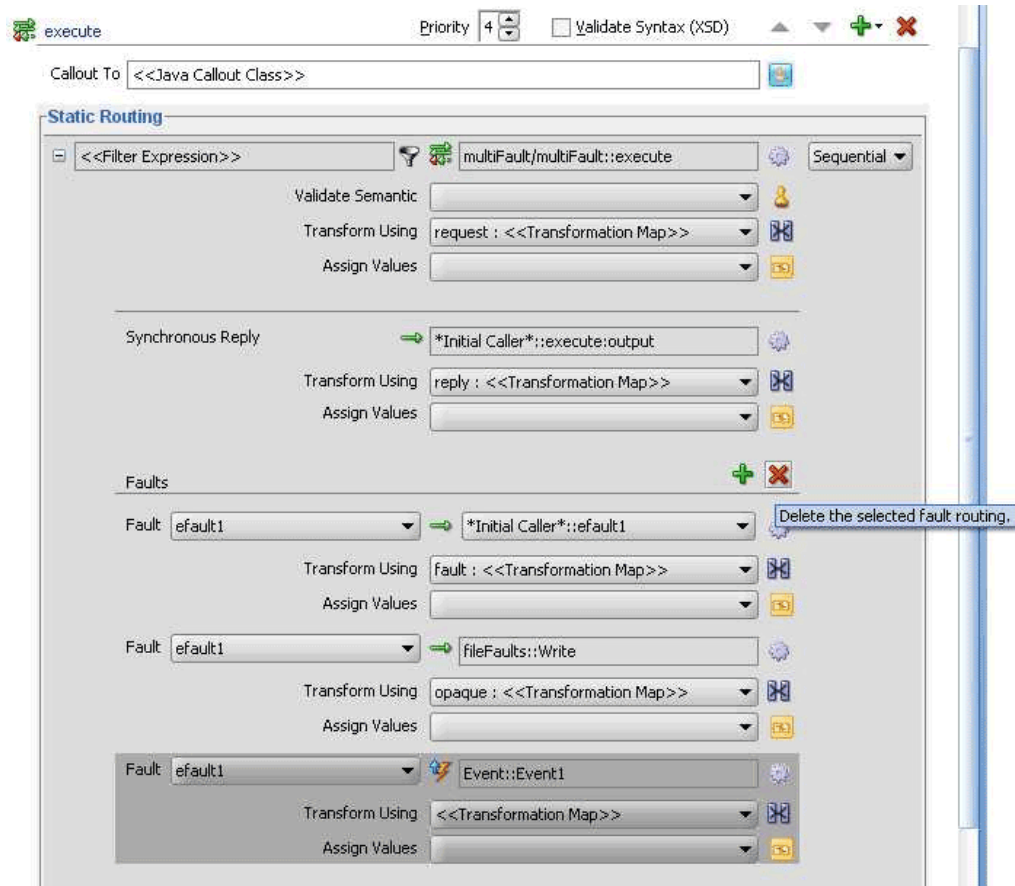
Note: You can route the same fault to multiple targets using different transformations.

To remove a fault routing section:

The following steps are performed in the Routing Rules section of the Mediator Editor.

- Highlight the fault routing you want to remove by clicking in the target service field, and then click **Delete the selected fault routing**, as shown in [Figure 19–12](#).

Figure 19–12 *Deleting a Fault Routing*



19.2.2.7 How to Specify an Expression for Filtering Messages

The filter expression routing rule lets you filter messages based on their payload. If the filter expression for a given message instance evaluates to true, the message is delivered to the target service or event specified within the routing rule.

For example, you route your data to customers in two different countries, such as US and Canada, but you only want notices regarding the MOBILE product line to be sent to US customers and the LANDLINE product line to customers in Canada. To implement this routing, you must define a routing rule for each component and operation pair that sends messages to the target customers. In addition, you specify filter expressions for the routing rules that send messages to the customers in the US or Canada.

You can also define filter expression message properties or message headers.

Filter Expression Message Properties

Two examples of filter expression message properties are shown in [Example 19–1](#).

Example 19–1 Filter Expression Message Properties

```
$in.property.custom.Priority = '1'
```

```
$in.property.tracking.ecid = '2'
```

Filter Expression Message Headers

Two examples of filter expression message headers are shown in [Example 19–2](#).

Example 19–2 Filter Expression Message Headers

```
$in.header.wsse_Security/wsse:Security/Priority = '234'
```

```
$in.header.wsse_Security/wsse:Security/Priority = '234'
```

For the preceding filter expression message headers to work, you must add the attribute shown in [Example 19–3](#) to the root element of the `.mplan` file.

Example 19–3 Attribute to Add

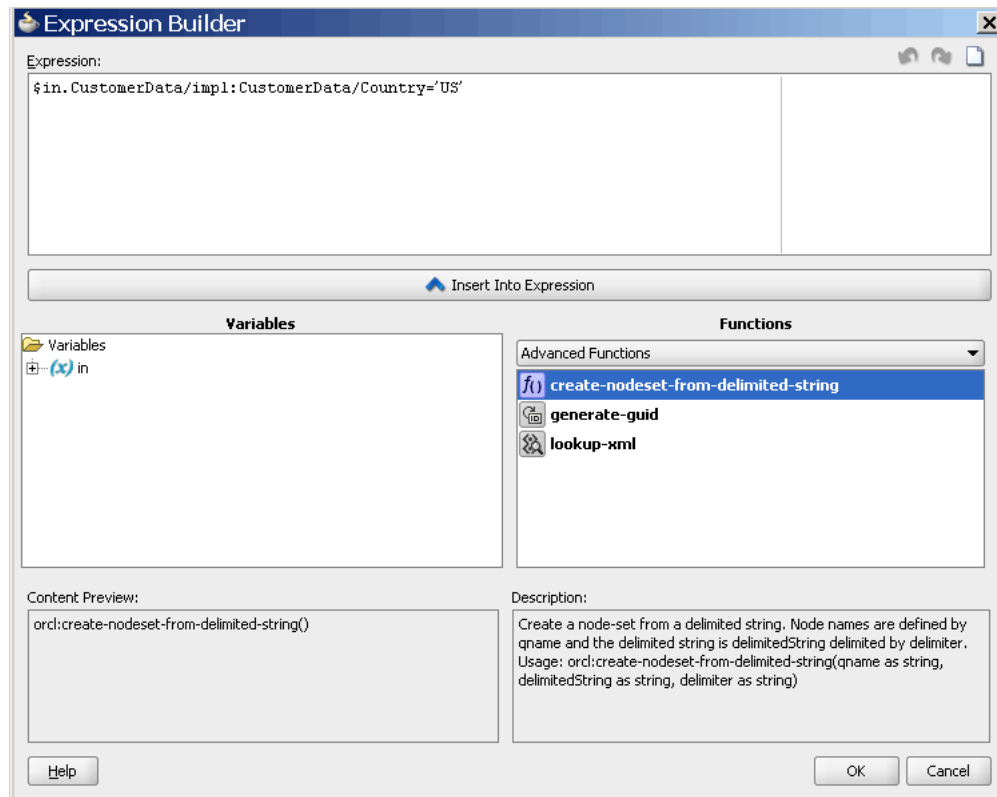
```
wsse = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
```

To specify an expression for filtering messages:

You can use the Expression Builder to graphically create a filter expression. The Expression Builder dialog contains the components and controls that assist you in designing a filter expression.

1. To the right of the **Filter Expression** field in the **Routing Rules** section, click the **Invoke Expression Builder** icon.

The Expression Builder dialog appears, as shown in [Figure 19–13](#).

Figure 19–13 Expression Builder Dialog

2. 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**

This field contains the actual expression used to filter messages. You can enter the filter expression either manually or by using the **Variable** field and the **Functions** palette.

Using the icons on the upper right side of this field, you can undo the last edit made, redo the last edit made, or clear the entire **Expression** field.

- **Variables**

This field contains the message defined for an Oracle Mediator component. Oracle JDeveloper parses the Oracle Mediator WSDL file and presents the message definition in the **Variables** field. The input message is stored in the `$in` variable, and you can use the `$in.properties` to access the properties of an input message.

If the input message consists of multiple parts, then you can use `$in.partname` to access a part of an input message.

- **Functions Palette**

This list provides a list of functions that you can include in an expression. When you select a function, a preview of how that function appears when

added to the **Expression** field appears in the **Content Preview** field, and a description of the function appears in the **Description** field.

- **Content Preview**

This field indicates how a value selected from the **Variables** field or **Functions** palette appears when it is inserted into the **Expression** field.

- **Description field**

This field describes the value selected from the **Variables** field or **Functions Palette**.

To specify a filter expression on a message payload:

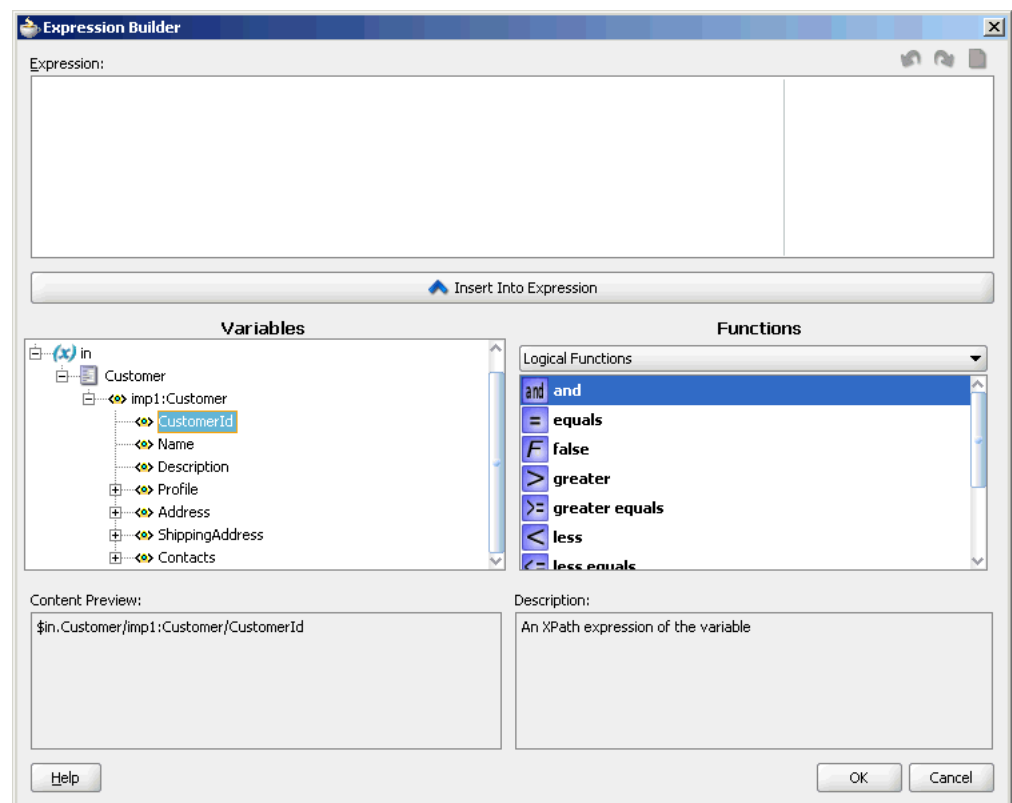
1. To the right of the **Filter Expression** field in the **Routing Rules** section, click the **Invoke Expression Builder** icon.

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.

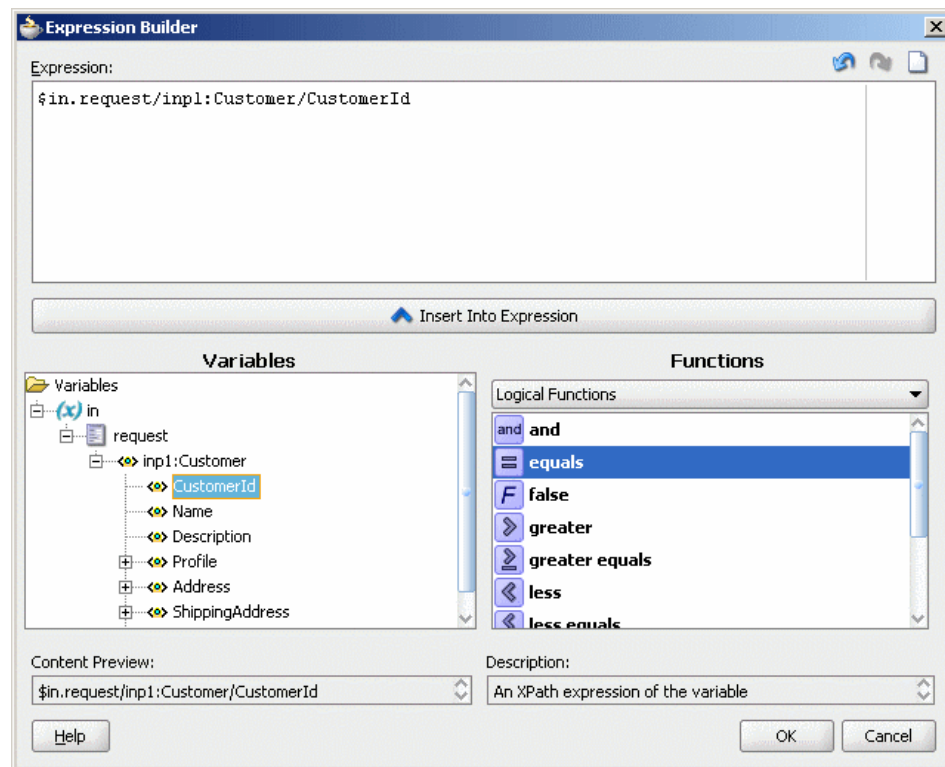
For example, the **CustomerId** element is shown selected in [Figure 19–14](#).

Figure 19–14 Expression Builder Dialog – Variables Element Selected



3. Click **Insert Into Expression**.

The expression is added in the **Expression** field, as shown in [Figure 19–15](#).

Figure 19–15 Expression Builder Dialog – Variables Element Inserted

4. From the **Functions** list, select the function to apply to the message payload. For example, **equals**.

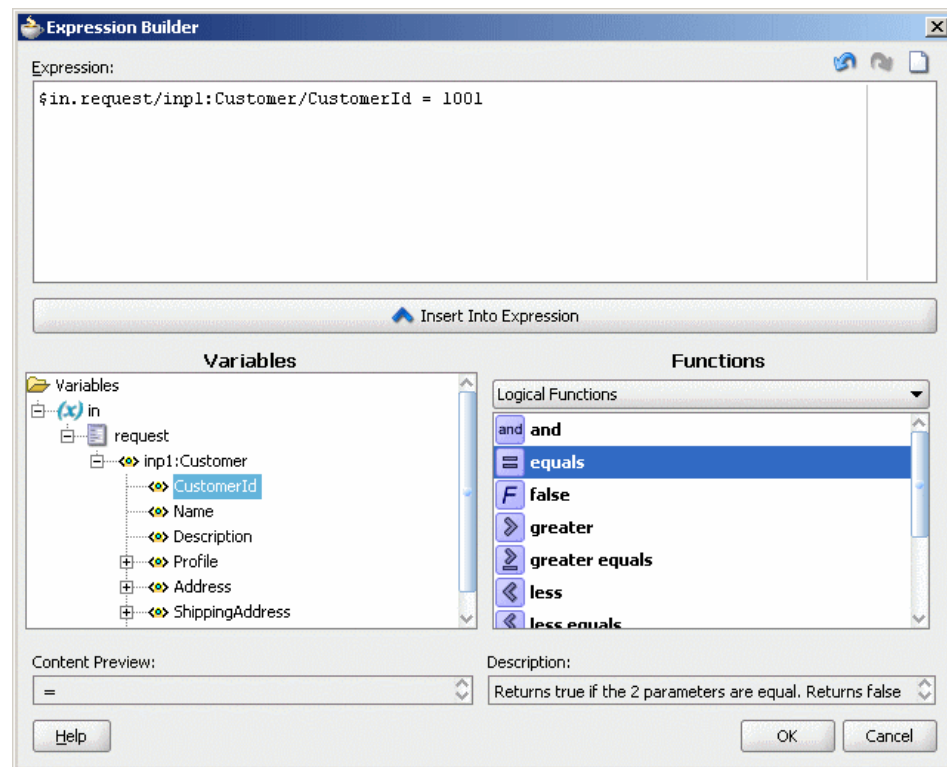
Functions are grouped in categories that are listed when you click the down arrow in the **Functions** list. For example, if you click the down arrow and select **Logical Functions**, the list appears as shown in [Figure 19–15](#).

5. Click **Insert Into Expression**.

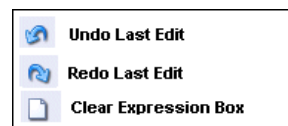
The XPath expression for the selected function is inserted into the **Expression** field.

6. Complete the expression.

In this example, the Customer ID must equal1001 to evaluate to true, as shown in [Figure 19–16](#).

Figure 19–16 Sample Expression Builder Dialog – Value Entered

7. If there are any errors, you can edit the expression manually, or use the expression editing icons, which are summarized in [Figure 19–17](#).

Figure 19–17 Expression Editing Icons

8. Click **OK**.

The expression is added to the **Routing Rules** section.

To modify or delete a filter expression, double-click the **Add Filter Expression** icon, and then modify or delete the expression in the **Expression** field of the Expression Builder.

19.2.2.8 How to Create Transformations

Oracle JDeveloper provides an XSLT Mapper that lets you specify a mapper file (XSL file) to transform data from one XML schema (expressed as an XSD file) to another. The XSLT Mapper enables data interchange among applications using different schemas. For example, you can map an incoming purchase order schema to an outgoing invoice schema. After you define an XSL file, you can reuse it in multiple routing rule specifications.

To create a transformation:

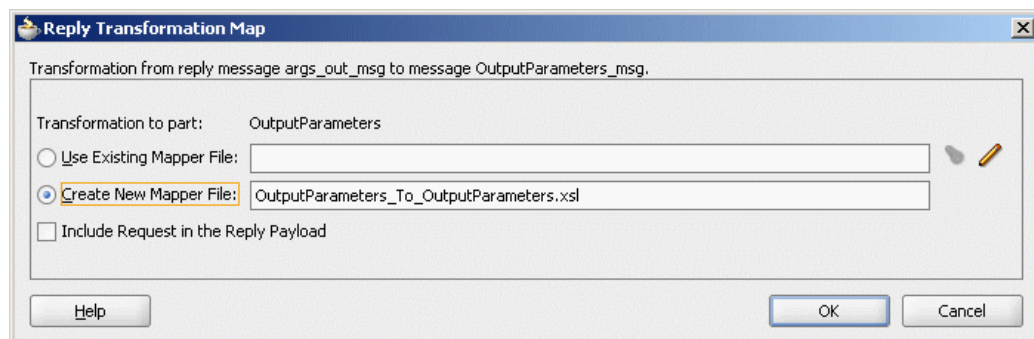
1. In the **Routing Rules** section, click the **Select an existing mapper file or create a new one** icon to the right of the **Transform Using** field.

The Request Transformation Map dialog appears. You can select an existing XSL file or create a new XSL file with the XSLT Mapper to perform the required transformation.

2. Do one of the following:
 - If the mapping file exists, select **Use Existing Mapper File** and then click **Browse** to find and select the mapper file to use.
 - To create a mapper file, select **Create New Mapper File**, and then enter the input information.
3. Repeat the above steps for any synchronous reply, callback, response, or fault messages.

In case of synchronous reply or fault message, the Reply Transformation Map dialog or the Fault Transformation Map dialog contains an **Include Request in the Reply Payload** option, as shown in [Figure 19–18](#).

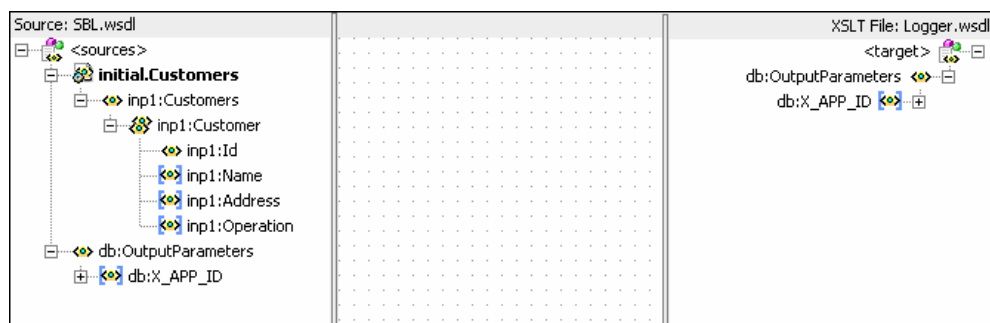
Figure 19–18 Reply Transformation Map Dialog



4. To create an `$initial` variable that contains the original message of a synchronous interaction, select the **Include Request in the Reply Payload** option.

The variable is created, as shown in [Figure 19–19](#).

Figure 19–19 Initial Variable in XSL File



Note: An initial message can also consist of multiple parts. You can use `$initial.partname` to access a part of the initial message.

If the parts of the inbound and outbound messages are identical, then no transformation is required for data interchange.

For information about the XSLT Mapper, see [Chapter 37, "Creating Transformations with the XSLT Mapper."](#)

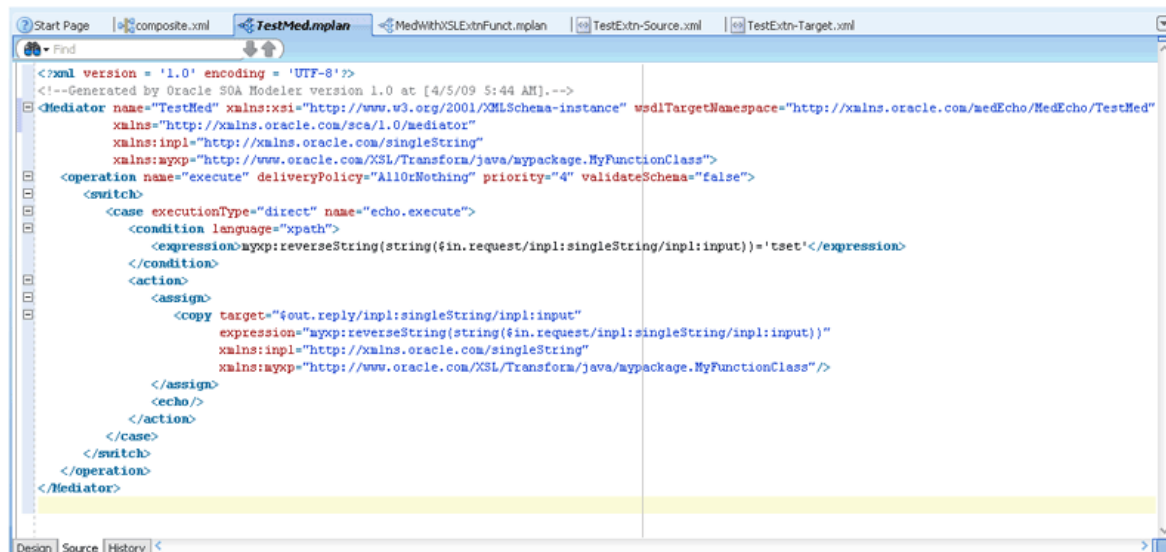
To add user-defined extension functions:

You can use the Expression Builder to include user-defined extension functions.

1. Create an XPath function.
2. Register the Jaxen XPath function with an Oracle Mediator service component in the `xpath-function.xml` file on the server.
3. Start Oracle JDeveloper.
4. Use the Expression Builder to customize the expression.
5. Deploy the Oracle JDeveloper project to Oracle WebLogic Server.
6. Copy the JAR file containing the user-defined extension functions to the `$BEAHOME/user_projects/domains/soainfra/autodeploy/soa-infra/APP-INF/lib` directory.
7. Modify the `.mplan` file of the project as follows:
 - Add the function namespace you defined for the extension functions under the Mediator element.
 - Add the function names under the Expression element.

This is shown in [Figure 19–20](#).

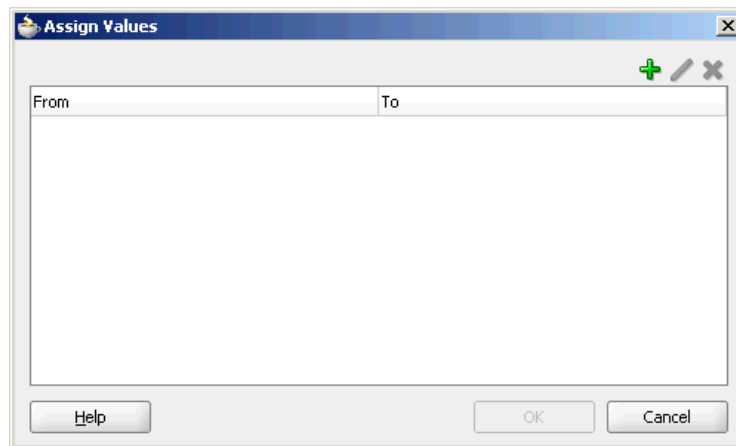
Figure 19–20 Project .mplan file – Modified to Use User-Defined Extension Functions



8. Invoke the test page with a suitable payload.

19.2.2.9 How to Assign Values

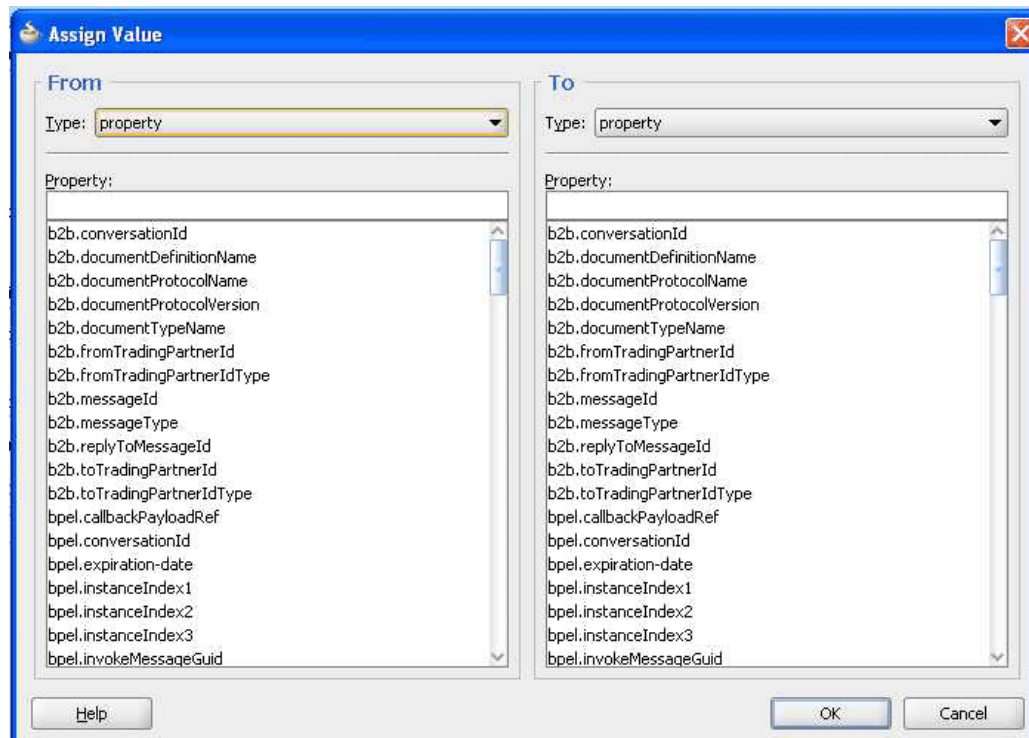
You can use the **Assign Values** field to propagate the headers, payload, and properties of a message from source to target. [Figure 19–21](#) shows the Assign Values dialog that is displayed when you click the **Assign Values** icon in the **Routing Rules** section.

Figure 19–21 Assign Values Dialog

To set the properties of the target message:

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

The Assign Value dialog is displayed, as shown in [Figure 19–22](#).

Figure 19–22 Assign Value Dialog

2. In the **From** section, select any of the following options from the **Type** list:
 - **Property:** Select this option to assign a 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 a value of an expression to the target message. When you click the **Invoke Expression Builder** icon to the right of

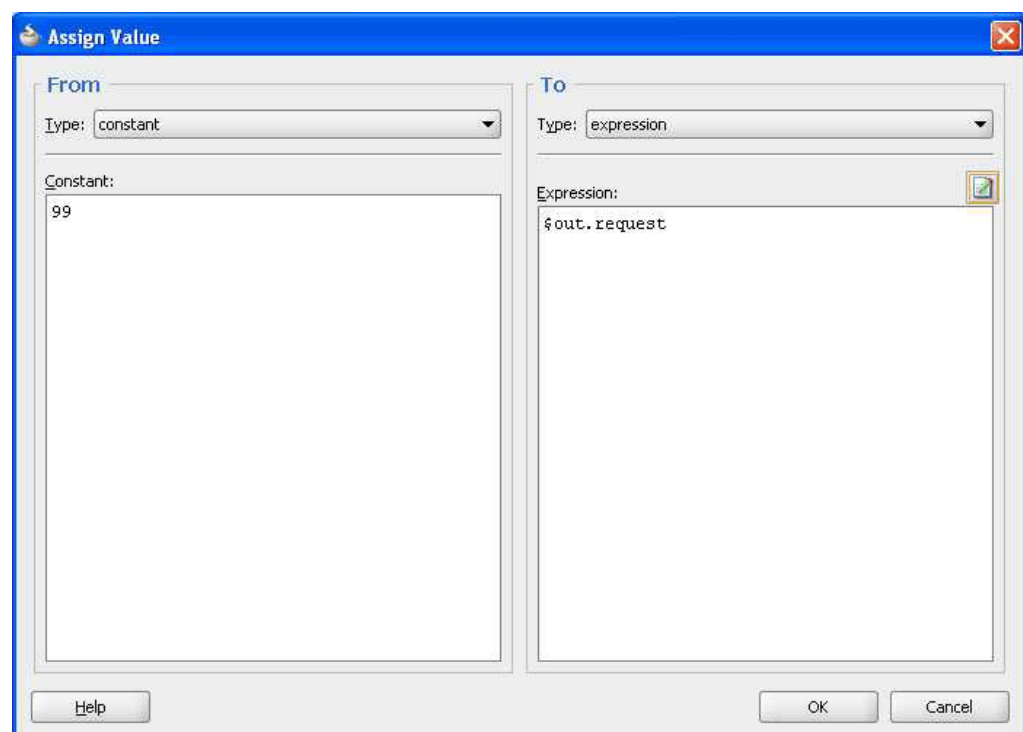
the **Expression** field, the Expression Builder dialog similar to the one shown in [Figure 19-13](#) is displayed.

For more information about the Expression Builder dialog, see [Section 19.2.2.7, "How to Specify an Expression for Filtering Messages."](#)

- **Constant:** Select this option to assign a constant value to the target message.
3. In the **To** section, select any of the following options:
- **Property:** Select this option to copy the value to a message property. The **Variable** field of the Expression Builder dialog contains an **\$out** variable that contains the output message. You can use **\$out.property** to access properties of an output message.
 - **Expression:** Select this option to copy the value to an expression. When you click the **Invoke Expression Builder** icon to the right of the **Expression** field, the Expression Builder dialog is displayed. The **Variable** field of the Expression Builder dialog contains an **\$out** variable that contains the output message. You can use **\$out.partname** to access a complete output message or part of an output message.

[Figure 19-23](#) shows a sample Assign Value dialog in which a constant value is specified as an expression.

Figure 19-23 Populated Assign Value Dialog



4. Click **OK** in the Assign Value dialog.
5. Click **OK**. The expression is added to **Assign Values** field of the **Routing Rules** section.

Notes:

- When you assign values to a particular Oracle Mediator property during event publishing, the assigned value does not get propagated to the subscribing event.
You can work around this issue by using transformations to include the property as part of the event body.
- You cannot assign values to the `jca.db.userName` and `jca.db.password` properties on Oracle WebLogic Server because their data sources do not support setting the user name or password dynamically to the `getConnection` method.

Table 19–1 through Table 19–3 list the various possibilities of assignment on constants and properties, payloads, and headers of a message from source to target.

Table 19–1 Possibilities on Constants and Properties

Source	Target	Example
Property	Property	<code><copy expression="\$in.property.jca.file.FileName" target="\$out.property.jca.file.FileName"/></code>
Constant	Property	<code><copy value="ConstantNameAssigned.xml" target="\$out.property.jca.file.FileName"/></code>

Table 19–2 Possibilities on Payload

Source	Target	Example
XPath Expression	Property	<code><copy expression="concat('ExprPropMed','-',oraext:generate-guid())" target="\$out.property.jca.file.FileName" xmlns:oraext="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc"/></code>
XPath Expression (below part level)	Property	<code><copy expression="\$in.body/impl:request/ProductReq/Make" target="\$out.property.jca.file.FileName" xmlns:impl="http://xmlns.oracle.com/psft"/></code>
Property	XPath Expression (below part level)	<code><copy value="\$in.property.jca.file.FileName" target="\$out.request/impl:request/ProductReq/Model" xmlns:impl="http://xmlns.oracle.com/psft"/></code>
Constant	XPath Expression (below part level)	<code><copy value="ConstantModel" target="\$out.request/impl:request/ProductReq/Model" xmlns:impl="http://xmlns.oracle.com/psft"/></code>
XPath Expression	XPath Expression	<code><copy expression="\$in.body" target="\$out.request"/></code>

Table 19–2 (Cont.) Possibilities on Payload

Source	Target	Example
XPath Expression (below part level)	XPath Expression (below part level)	<pre><copy expression="\$in.body/impl:request/ProductRe q/Make" target="\$out.request/impl:request/ProductRe q/Model" xmlns:impl="http://xmlns.oracle.com/psft"/></pre>

Table 19–3 Possibilities on Header

Source	Target	Example
XPath Expression (below part level)	Property	<pre><copy expression="\$in.header.inpl_ header/inpl:header/Name" target="\$out.property.jca.file.FileName" xmlns:inpl="http://xmlns.oracle.com/psft"/></pre>
Property	XPath Expression (below part level)	<pre><copy value="\$in.property.jca.file.FileName" target="\$out.header.inpl_ header/inpl:header/Name" xmlns:inpl="http://xmlns.oracle.com/psft"/></pre>
Constant	XPath Expression (below part level)	<pre><copy value="NewID.xml" target="\$out.header.inpl_ header/inpl:header/Id" xmlns:inpl="http://xmlns.oracle.com/psft"/></pre>
Constant	XPath Expression (below part level)	<pre><copy value="sampleusername" xmlns:wssel="http://docs.oasis-open.org/wss /2004/01/oasis-200401-wss-wssecurity-secext -1.0.xsd" target="\$out.header.wssel_ Security/wssel:Security/wssel:UsernameToken /wssel:Username"/></pre>
XPath Expression	XPath Expression	<pre><copy target="\$out.header.inpl_header" expression="\$in.header.inpl_header" xmlns:inpl="http://xmlns.oracle.com/psft"/></pre>
XPath Expression (below part level)	XPath Expression (below part level)	<pre><copy target="\$out.header.inpl_ header/inpl:header/Name" expression="\$in.header.inpl_ header/inpl:header/Id" xmlns:inpl="http://xmlns.oracle.com/psft"/></pre>

19.2.2.10 What You May Need to Know About the Assign Activity

Note the following issues about the assign activity.

- The assign activity is executed in the order that is present in the Oracle Mediator `<copy>` element.
- A source XPath expression should always refer to a leaf node while the source is assigned to a target property. Otherwise, all the values of the child nodes in the source get concatenated and are assigned to the target property. [Example 19–4](#) provides details.

Example 19–4 XPath Expression Referring to a Leaf Node

```
<copy target="$out.property.jca.file.FileName"
expression="$in.body/impl:request/ProductReq/Make"
xmlns:impl="http://xmlns.oracle.com/psft"/>
```

Note: A leaf node is a node with no child nodes.

- While assigning a constant or a property to a target XPath expression, the target XPath expression should always point to a leaf node. Otherwise, nonleaf nodes contain only a string value that may generate nonvalid XML according to the .xsd file. [Example 19–5](#) provides details.

Example 19–5 Target XPath Expression Pointing to a Leaf Node

```
<copy target="$out.request/inpl:request/ProductReq/Make" value="NewMakeValue"
xmlns:inpl="http://xmlns.oracle.com/psft"/>
```

In this example, `$out.request/inpl:request/ProductReq/Make` refers to the leaf node.

- If a transformation is available, then while assigning a source part to a target part, the target is overwritten because the assign activity occurs on top of the transformation. If the transformation is not available, then the assign activity creates the target. [Example 19–6](#) provides details.

Example 19–6 Transformation Availability and Assign Activity

```
<copy target="$out.request" expression="$in.body"/>
```

```
<copy target="$out.header.inpl_header" expression="$in.header.inpl_header"
xmlns:inpl="http://xmlns.oracle.com/psft"/>
```

- If one of the child nodes in the target payload has to be modified, then there are the following two use cases:
 - If a transformation is available, then directly assign a source expression to a target XPath expression that is pointing to that child node in the target. [Example 19–7](#) provides details.

Example 19–7 Direct Assignment of a Source Expression to a Target XPath Expression

```
<copy value="ConstantModel"
target="$out.request/inpl:request/ProductReq/Model"
xmlns:inpl="http://xmlns.oracle.com/psft"/>
```

- If a transformation is not available, then there are two steps involved. First, assign the source part to the target part, and then assign the source expression to a target XPath expression that is pointing to the child node in the target. [Example 19–8](#) provides details.

Example 19–8 Assignments if Transformations are Unavailable

```
<copy target="$out.request" expression="$in.body"/> and <copy
value="ConstantModel" target="$out.request/inpl:request/ProductReq/Model"
xmlns:inpl="http://xmlns.oracle.com/psft"/>
```

- When only one of the child nodes of the source has to be propagated into a target, then first ensure that there is no transformation invoked. Then, assign the source XPath expression to point to the required child node. [Example 19–9](#) provides details.

Example 19–9 One Child Node of the Source is Propagated into a Target

```
<copy target="$out.request/impl:ProductReq"
  expression="$in.body/impl:request/ProductReq"
  xmlns:impl="http://xmlns.oracle.com/psft"/>
```

In this case, the source element evaluated from `$in.body/impl:request/ProductReq` does not contain a complete tree structure that starts from the root element, but contains only a child node. [Example 19–10](#) provides details.

Example 19–10 Structure Starting from the Root Element that Contains Only a Child Node

```
<ProductReq>
  <Make>MAKE</Make>
  <Model>MODEL</Model>
</ProductReq>
```

- If there are multiple assign activities in an Oracle Mediator and each source XPath expression points to a different child node, then there are the following two use cases:
 - If a transformation is available, then the corresponding child node in the target is updated.
 - If a transformation is not available, then the target should be a multiple part target with each part referring to the source child node.
- With headers, if the `passThroughHeader` property is set, then
 - Any header manipulation in a transformation is updated in the target headers.
 - The part level assign activity overwrites the target header part.
 - The below part level node assign activity updates the corresponding node in the target.
- If multiple source nodes (below part level) are assigned to the same target node (below part level), then the target node contains the value of the last `copy` element in the assign activity. [Example 19–11](#) provides details.

Example 19–11 Multiple Source Nodes Assigned to the Same Target Node

```
<copy target="$out.request/impl:request/ProductReq/Make"
  expression="$in.body/impl:request/ProductReq/Model"
  xmlns:impl="http://xmlns.oracle.com/psft"/>

<copy target="$out.request/impl:request/ProductReq/Make"
  expression="$in.body/impl:request/Description"
  xmlns:impl="http://xmlns.oracle.com/psft"/>
```

In [Example 19–11](#), the first `copy` element does not have any effect because the second `copy` element overwrites it.

- If the XPath expression results in a list (multiple occurrences), then there are the following two use cases:
 - If the list contains a single element, then the XPath expression is propagated.
 - If the list contains multiple elements, then the XPath expression is not supported.

- The following activities happen while assigning a source child node to a target child node:
 1. The source child node name and namespace are overwritten by the target node name and namespace, respectively.
 2. The target child node is replaced by the source child node in the parent node of the target node.

19.2.2.11 How to Access Headers for Filters and Assignments

When the Expression Builder is invoked from an Oracle Mediator, either for defining a filter or for defining an assignment source or target, the WSDL file is parsed. This automatically detects any SOAP headers for the current routing rule operation and makes them visible as variables under the `in` or `out` folder as `header./ns_elementName/`, as shown in [Figure 19-24](#). Here, `ns` is the namespace prefix and `elementName` is the root element name for the header schema.

The following examples provide details.

Example 1: Namespace Prefixes `wsse` and `ns1` Are Already Defined

Assume the namespace prefixes `wsse` and `ns1` are already defined in the WSDL file or the `.mplan` file. You can then write an XPath expression as follows:

```
$in.header.wsse_Security/wsse:Security/ns1:Foo/Priority
```

Example 2: Schema Without a Namespace Predefined in the WSDL File

Assume you want to use a schema that does not have a namespace predefined in the WSDL file. The Expression Builder is then enhanced to allow you to enter `{full_namespace}` instead of a prefix. The Expression Builder then generates a unique prefix and the prefix definition is added to the `.mplan` file.

For example, enter the expression in the Expression Builder shown in [Example 19-12](#):

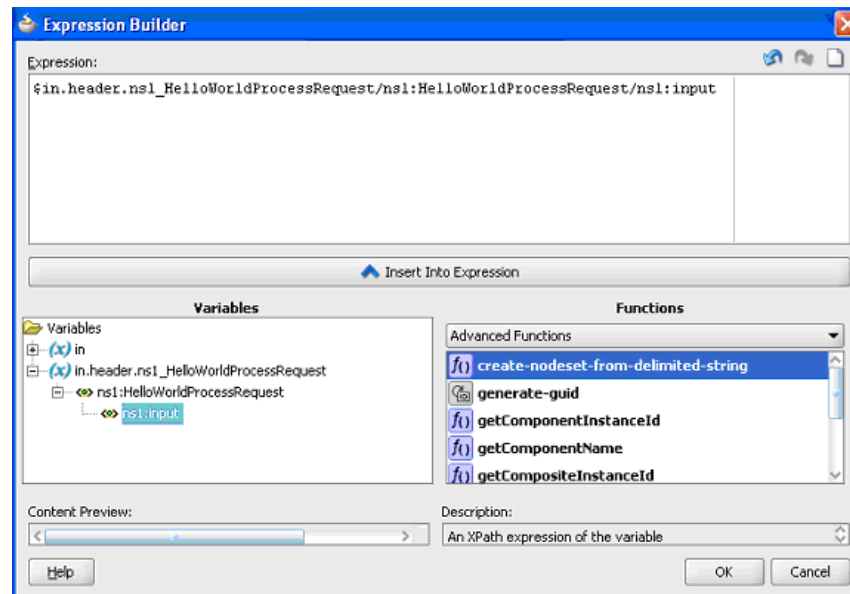
Example 19-12 Expression

```
$in.header.{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd}_Security/
{"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"}:
Security/{"http://www.globalcompany.com/ns/OrderBooking"}:Foo/Priority
```

The `.mplan` file contains the content shown in [Example 19-13](#).

Example 19-13 Contents of `.mplan` File

```
xmlns:ns1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns:ns2="http://www.globalcompany.com/ns/OrderBooking"
...
expression="$in.header.ns1_Security/ns1:Security/ns2:Foo/Priority"
```


Figure 19–24 Expression Builder Dialog - Automatic Header Detection

By default, SOAP headers are not passed through by Oracle Mediator. You must add the `passThroughHeader` endpoint property to the corresponding Oracle Mediator routing service:

```
<property name="passThroughHeader">true</property>
```

For example, to add this property, you can modify the `composite.xml` file, as shown in [Example 19–14](#).

Example 19–14 `passThroughHeader` Property

```
<component name="Mediator1">
  <implementation.mediator src="Mediator1.mplan" />
  <property name="passThroughHeader">true</property>
</component>
```

For the headers to pass through, the source and the target must have the same QName (name and namespace). If the source and the target have different QNames, then either a transformation or part-level assignment must be performed.

It is important to note that, with a `passthrough` Oracle Mediator (without a transformation or assign), if the source and target part QNames are not identical, then Oracle Mediator passes through the message payloads to the target service without any error. However, this can result in an error in the target service because the message payloads are not reconstructed according to the message structure of the target service.

Notes:

- The user interface supports both SOAP 1.1 and SOAP 1.2.
 - For automatic header detection, a concrete WSDL file must be used when creating the Oracle Mediator service component.
 - Assignments execute after filters. Therefore, if you are assigning a value in a custom header, then the particular assignment is not visible to the filter.
-

19.2.2.11.1 Manual Expression Building for Accessing Headers for Filters and Assignments

There are use cases in which the header schemas cannot be determined from the WSDL files. For example, security headers that are appended to a message, or the headers for an Oracle Mediator that are created using an abstract WSDL file. To access these headers, you must manually enter the XPath expression into the Expression Builder.

The syntax for header expressions is shown in [Example 19–15](#).

Example 19–15 Header Expressions Syntax

```
$in.header.<header root element namespace prefix>_<header root element name>/<xpath>
```

Therefore, for the header shown in [Example 19–16](#).

Example 19–16 Header Syntax

```
<wsse:Security
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
ext-1.0.xsd">
  <Priority>234</Priority>
</wsse:Security>
```

The filter expression is as follows:

```
$in.header.wsse_Security/wsse:Security/Priority = '234'
```

The assignment expression is as shown in [Example 19–17](#).

Example 19–17 Assignment Expression

```
<copy target="$out.property.jca.jms.priority"
expression="$in.header.wsse_Security/wsse:Security/Priority"/>
```

For the preceding expressions to work, you must add the attribute shown in [Example 19–18](#) to the root element of the `.mplan` file.

Example 19–18 Addition of Attribute to `.mplan` File

```
wsse = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
```

19.2.2.11.2 Manual Expression Building for Accessing Properties for Filters and Assignments

An example of a filter expression is as follows.

```
$in.property.tracking.ecid = '2'
```

An example of an assignment expression is as follows.

```
<copy target="$out.property.tracking.ecid" value="$in.property.tracking.ecid"/>
```

19.2.2.12 How to Use Semantic Validation

You can specify Schematron files for validating an inbound message and its various parts. Schematron version 1.5 is the supported version.

Perform the following steps for specifying a Schematron schema to validate an inbound message and its various parts.

To use semantic validation:

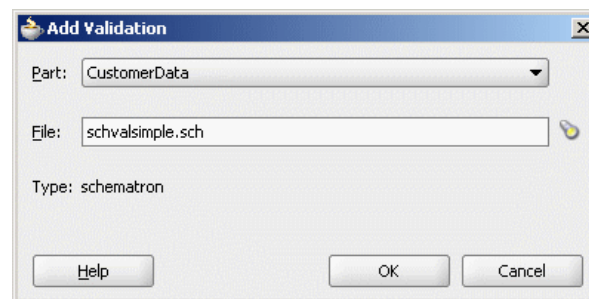
1. To the right of the **Validate Semantic** field, click the **Select Validation File** icon.
The Validations dialog is displayed.
2. Click **Add**.
The Add Validation dialog is displayed.
3. From the **Part** list, select a message part.
4. To the right of the **File** field, click **Search**.
The SOA Resource Browser dialog is displayed.
5. Select a Schematron file and click **OK**.

Notes:

- Schematron files usually have a `.sch` extension.
- No error message or warning is displayed if the selected Schematron file is empty.

The Add Validation dialog is updated, as shown in [Figure 19-25](#).

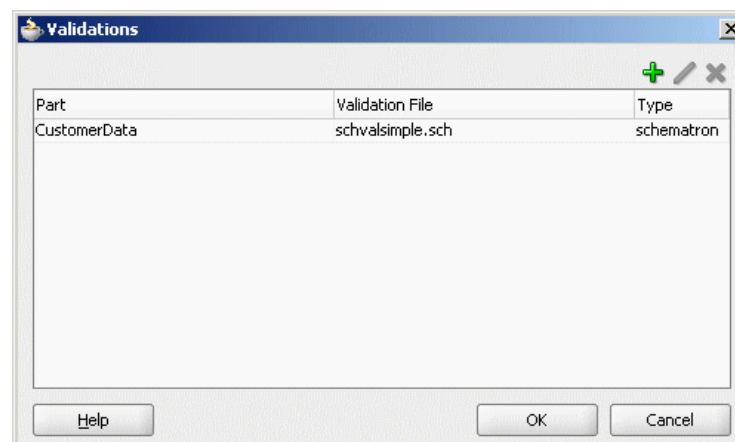
Figure 19-25 Add Validation Dialog



6. Click **OK**.

The Validation dialog is updated, as shown in [Figure 19-26](#).

Figure 19-26 Validation Dialog



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

For more information about building a Schematron schema, see the resources available at

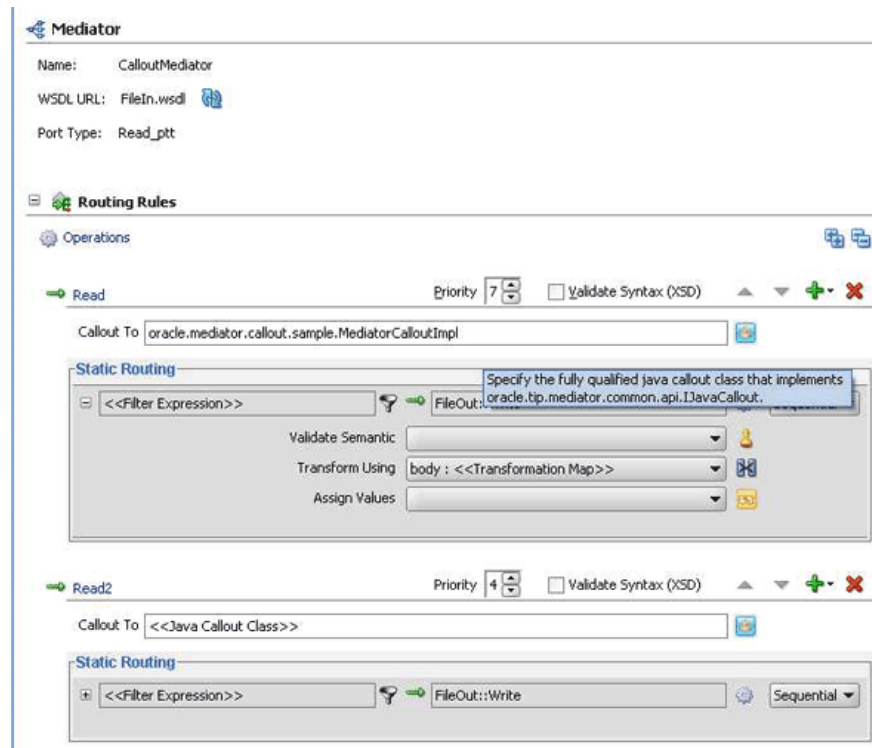
<http://www.schematron.com>

Note: In semantic validation, if you check for the length of each element name, then the element name may change for a different set of inputs. This happens when there are white spaces between nodes because the parser treats the white spaces as test nodes.

19.2.2.13 How to Use Java Callouts

Java callouts enable you to use external Java classes to manipulate messages flowing through the Oracle Mediator. Only one Java callout is supported per operation or event subscription. The callout class must implement the `oracle.tip.mediator.common.api.IJavaCallout` interface. Callouts are available for both static and dynamic routings. Figure 19–27 shows a sample Oracle Mediator with two operations, in which both the operations have one routing rule each and the first operation has a callout class.

Figure 19–27 Sample Oracle Mediator Supporting Java Callout



To make Java callout classes available:

You must ensure that the Java callout class is available on the server. You can use any of the following methods for this:

- Copy the Java class to the `SCA-INF/classes` folder.
- Copy the JAR file containing the Java class to the `SCA-INF/lib` folder.

- Copy the JAR file containing the Java class to the `$DOMAIN_HOME/lib` folder.

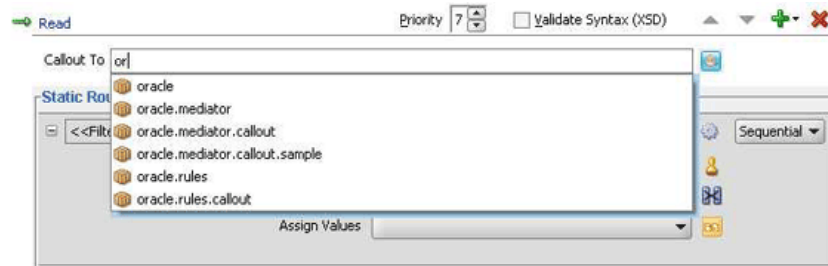
If you want to make the Java callout class available to multiple Oracle Mediators, copy the JAR file containing the Java class to the `$DOMAIN_HOME/lib` folder.

To enter the Java class for the callout:

You can either manually enter the Java class or select a class from the Class Browser.

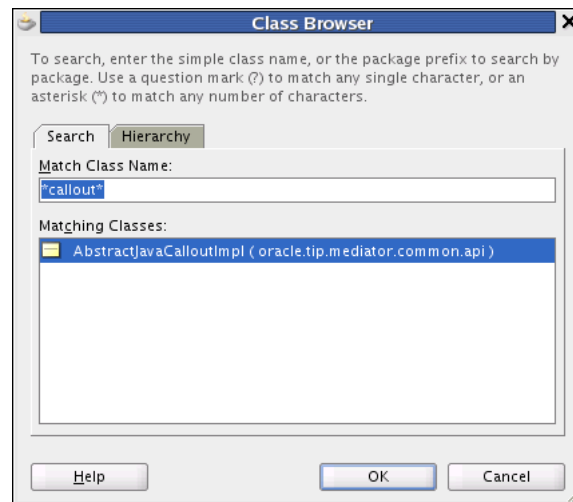
- To manually enter the name of the Java callout class, start typing the class name in the **Callout To** field, as shown in [Figure 19–28](#). The auto-completion feature of Oracle JDeveloper completes the address and the classes in the current project.

Figure 19–28 Callout To Field



- To select from a list of available classes, click the **Select Java Callout Class** icon. The standard Oracle JDeveloper class browser appears, as shown in [Figure 19–29](#).

Figure 19–29 Class Browser Dialog



The class browser is filtered so it only displays classes that implement the `oracle.tip.mediator.common.api.IjavaCallout` interface.

To set the payload root element (when using a filter expression):

If you have a Java callout in Oracle Mediator and use a filter expression in the same Oracle Mediator, you must set the root element for the payload, as shown in [Example 19–19](#).

Example 19–19 Setting the Root Element for the Payload

```

changexml doc = XmlUtils.getXmlDocument(ChangedDoc);
String mykey = "request";
message.addPayload(mykey, changexml doc.getDocumentElement());

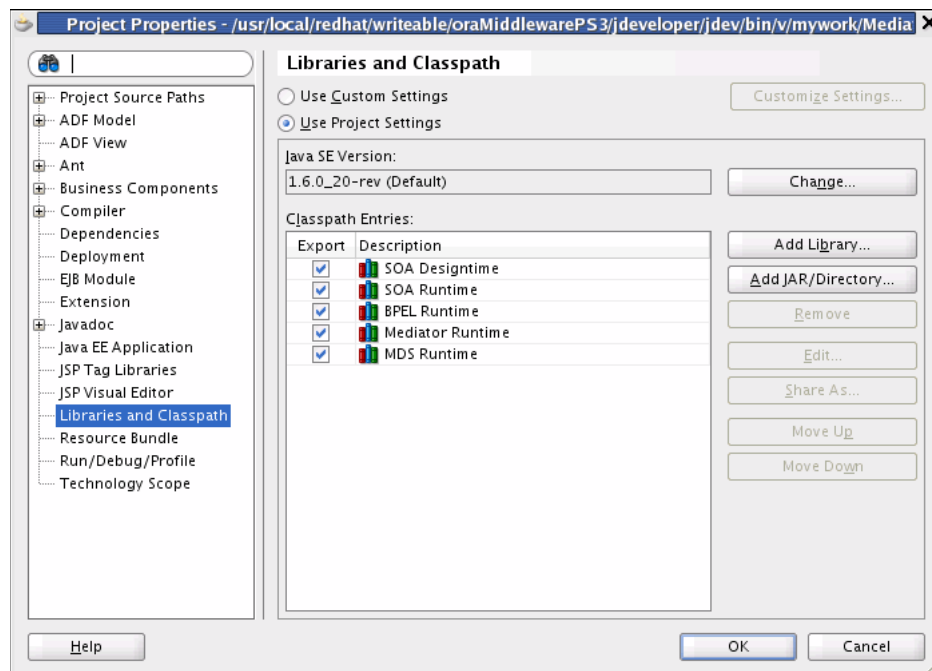
```

To enable domain value map and cross reference functions:

To use domain value map functions or cross reference functions in a Java callout, you must add the `soa-xpath-exts.jar` file to the project and import the necessary Java classes into your code.

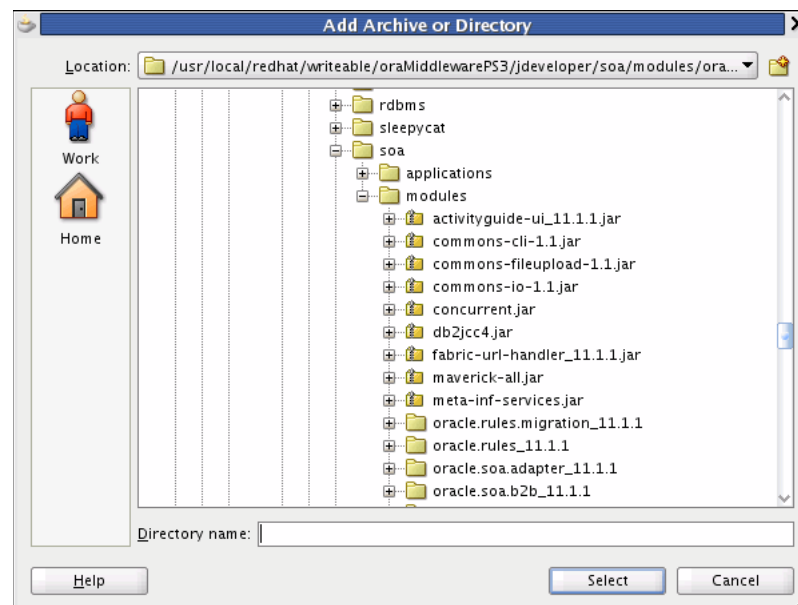
1. In the Oracle JDeveloper Projects Explorer, right-click the name of the project containing the Java callout.
2. Select **Project Properties**.
The Project Properties dialog appears.
3. In the left panel, select **Libraries and Classpath**, as shown in [Figure 19–30](#).

Figure 19–30 Libraries and Classes on the Project Properties Dialog



4. Click **Add JAR/Directory**.

The **Add Archive or Directory** dialog appears, as shown in [Figure 19–31](#).

Figure 19–31 Add Archive or Directory Dialog

5. In the explorer tree, expand the directories to select `<JDEV_HOME>/jdeveloper/soa/modules/oracle.soa.fabric_11.1.1/soa-xpath-exts.jar`, and then click **Select**.

The JAR file appears in the Classpath Entries list.

6. Click **OK**.

Notes: When using domain value map functions, import the following into your Java class:

- `oracle.tip.dvm.LookupValue`
- `oracle.tip.dvm.exception.DVMException`

When using cross reference (xref) functions, import the following into your Java class:

- `oracle.tip.xref.xpath.XRefXPathFunctions`
 - `oracle.tip.xref.exception.XRefException`
-

Oracle Mediator Java Callout API

The Java callout API defines two interfaces:

`oracle.tip.mediator.common.api.IjavaCallout` and
`oracle.tip.mediator.common.api.CalloutMediatorMessage`.

Table 19–4 lists and describes the methods in the `oracle.tip.mediator.common.api.IjavaCallout` interface.

Table 19–4 Description of Methods in the IjavaCallout Interface

Method	Description
<code>initialize</code>	This method is invoked when the callout implementation class is instantiated for the first time.

Table 19–4 (Cont.) Description of Methods in the *ljavaCallout* Interface

Method	Description
<code>preRouting</code>	This method is called before Oracle Mediator starts executing the cases. You can customize this method to include validations and enhancements.
<code>preRoutingRule</code>	This method is called before Oracle Mediator starts executing any particular case. You can customize this method to include case-specific validations and enhancements.
<code>preCallbackRouting</code>	This method is called before Oracle Mediator finishes executing callback handling. You can customize this method to perform callback auditing and custom fault tracking.
<code>postRouting</code>	This method is called after Oracle Mediator finishes executing the cases. You can customize this method to perform response auditing and custom fault tracking.
<code>postRoutingRule</code>	This method is called after Oracle Mediator starts executing the cases. You can customize this method to perform response auditing and custom fault tracking.
<code>postCallbackRouting</code>	This method is called after Oracle Mediator finishes executing callback handling. You can customize this method to perform callback auditing and custom fault tracking.

Note: If you change the message properties of an Oracle Mediator by using a Java callout in the `preRoutingRule` method or the `preRouting` method, then you must explicitly copy the changed property to the outbound message by using Oracle Mediator assignment functionality. For example, if you are changing the `jca.file.FileName` property in a Java callout, then you must update the Oracle Mediator assignment statement as follows:

```
<assign>
<copy target="$out.property.jca.file.FileName"
expression="$in.property.jca.file.FileName" />
</assign>
```

Table 19–5 discusses the methods in the `CalloutMediatorMessage` interface.

Table 19–5 Description of Methods in the *CalloutMediatorMessage* Interface

Method	Description
<code>addPayload</code>	This method sets a payload of the Oracle Mediator messages.
<code>addProperty</code>	This method adds a property to the Oracle Mediator messages.
<code>addHeader</code>	This method adds a header to the Oracle Mediator messages.
<code>getProperty</code>	This method retrieves Oracle Mediator message properties by providing the property name.
<code>getProperties</code>	This method retrieves Oracle Mediator message properties.
<code>getId</code>	This method retrieves the instance ID of the Oracle Mediator messages. This instance ID is the Oracle Mediator instance ID created for that particular message.
<code>getPayload</code>	This method retrieves a payload of the Oracle Mediator messages.
<code>getHeaders</code>	This method retrieves a header of the Oracle Mediator messages.

Table 19–5 (Cont.) Description of Methods in the CalloutMediatorMessage Interface

Method	Description
getComponentDN	This method retrieves a componentDN for the Oracle Mediator service component.

Notes:

- The `oracle.tip.mediator.common.api.AbstractJavaCalloutImpl` class is a dummy implementation¹ of the `IJavaCallout` interface. This class defines all the methods present in the `IJavaCallout` interface. Therefore, you can extend this class to override only a few specific methods of the `IJavaCallout` interface.
- Details of the processing occurring within the Java callout are not displayed in the Oracle Mediator audit trail screen.

¹ Dummy implementation of an interface means that the implementation class provides definitions for all the methods declared in the particular interface, but one or more defined methods may have an empty method body. Extending a dummy implementation class is much easier because you can choose to override only a subset of the methods, unlike implementing an interface and defining all the methods.

Sample Java Callout Class

[Example 19–20](#) shows a sample Java callout class:

Example 19–20 Sample Java Callout Class

```
package qa.as11tests.javacallout;

import com.collaxa.cube.persistence.dto.XmlDocument;

import com.oracle.bpel.client.NormalizedMessage;

import java.util.logging.Logger;
import java.util.Map;
import java.util.Iterator;

import oracle.tip.mediator.common.api.CalloutMediatorMessage;
import oracle.tip.mediator.common.api.ExternalMediatorMessage;
import oracle.tip.mediator.common.api.IJavaCallout;
import oracle.tip.mediator.common.api.MediatorCalloutException;
import oracle.tip.mediator.metadata.CaseType;
import oracle.tip.mediator.utils.XmlUtils;

import oracle.tip.pc.services.functions.ExtFunc;

import oracle.xml.parser.v2.XMLDocument;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

public class JavaCalloutSanity implements IJavaCallout {
    Logger logger = Logger.getLogger("Callout");
    public JavaCalloutSanity() { }
```

```

    public void initialize(Logger logger) throws MediatorCalloutException {
        this.logger = logger;
        this.logger.info("Initializing...");
    }

    public boolean preRouting(CalloutMediatorMessage calloutMediatorMessage) {
        System.out.println("Pre routing...");
        String sPayload = "null";
        String sPayload_org = "null";
        for (Iterator msgIt =
calloutMediatorMessage.getPayload().entrySet().iterator();
        msgIt.hasNext(); ) {
            Map.Entry msgEntry = (Map.Entry)msgIt.next();
            Object msgKey = msgEntry.getKey();
            Object msgValue = msgEntry.getValue();
            if (msgKey.equals("request"))
                sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
        }
        sPayload_org = sPayload;
        String tobeReplaced = "CHANGE_THIS";
        String replaceWith = "JAVA_CALLOUT_||_PRE_ROUTING";
        int start = sPayload.indexOf(tobeReplaced);
        StringBuffer sb = new StringBuffer();
        sb.append(sPayload.substring(0, start));
        sb.append(replaceWith);
        sb.append(sPayload.substring(start + tobeReplaced.length()));
        String changedPayload = sb.toString();
        String uid;
        try {
            uid = ExtFunc.generateGuid();
        } catch (Exception e) {
        }
        XMLDocument changedoc;
        try {
            changedoc = XmlUtils.getXmlDocument(changedPayload);
            String mykey = "request";
            calloutMediatorMessage.addPayload(mykey, changedoc);
            //calloutMediatorMessage.getPayload().put(mykey, changedoc);
        } catch (Exception e) {
        }
        System.out.println("Changed from : \n"+sPayload_
org+"\nTo\n"+changedPayload);
        System.out.println("End Pre routing...\n\n");
        return false;
    }

    public boolean postRouting(CalloutMediatorMessage calloutMediatorMessage,
        CalloutMediatorMessage calloutMediatorMessage1,
        Throwable throwable) throws
MediatorCalloutException {
        System.out.println("Start Post routing...");
        String sPayload = "null";
        String sPayload_org = "null";
        for (Iterator msgIt =
calloutMediatorMessage1.getPayload().entrySet().iterator();
        msgIt.hasNext(); ) {
            Map.Entry msgEntry = (Map.Entry)msgIt.next();
            Object msgKey = msgEntry.getKey();
            Object msgValue = msgEntry.getValue();
            if (msgKey.equals("reply"))
                sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
        }
    }

```

```

        sPayload_org = sPayload;
        String tobeReplaced = "POST_ROUTING_RULE_REQUEST_REPLY";
        String replaceWith = "POST_ROUTING_RULE_REQUEST_REPLY_|_|_POSTROUTING_|_|_
JAVA_CALLOUT_WORKING";
        int start = sPayload.indexOf(tobeReplaced);
        StringBuffer sb = new StringBuffer();
        sb.append(sPayload.substring(0, start));
        sb.append(replaceWith);
        sb.append(sPayload.substring(start + tobeReplaced.length()));
        String changedPayload = sb.toString();
        XMLDocument changedoc;
        try {
            changedoc = XmlUtils.getXmlDocument(changedPayload);
            String mykey = "reply";

calloutMediatorMessage1.addPayload(mykey,changedoc.getDocumentElement());
            // calloutMediatorMessage1.getPayload().put(mykey,
changedoc.getDocumentElement());
        } catch (Exception f) {
        }
        System.out.println("Changed from : \n"+sPayload_org+"\nTo\n"+
            changedPayload);
        System.out.println("End Post routing...\n\n");
        return false;
    }

    public boolean preRoutingRule(CaseType caseType,
        CalloutMediatorMessage calloutMediatorMessage) {
        System.out.println("\nStart PreRoutingRule.\n");
        String sPayload = "null";
        String sPayload_org = "null";
        for (Iterator msgIt =
            calloutMediatorMessage.getPayload().entrySet().iterator();
            msgIt.hasNext(); ) {

            Map.Entry msgEntry = (Map.Entry)msgIt.next();
            Object msgKey = msgEntry.getKey();
            Object msgValue = msgEntry.getValue();
            if(msgKey.equals("request"))
                sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
        }
        sPayload_org = sPayload;
        String tobeReplaced = "PRE_ROUTING";
        String replaceWith = "PRE_ROUTING_|_|_PRE_ROUTING_RULE";
        int start = sPayload.indexOf(tobeReplaced);
        StringBuffer sb = new StringBuffer();
        sb.append(sPayload.substring(0, start));
        sb.append(replaceWith);
        sb.append(sPayload.substring(start + tobeReplaced.length()));
        String changedPayload = sb.toString();
        XMLDocument changedoc;
        try {
            changedoc = XmlUtils.getXmlDocument(changedPayload);
            String mykey = "request";
            calloutMediatorMessage.addPayload(mykey,changedoc);
            // calloutMediatorMessage.getPayload().put(mykey, changedoc);
        } catch (Exception e) {
        }
        System.out.println("Changed from : \n"+sPayload_
org+"\nTo\n"+changedPayload);
    }

```

```
        System.out.println("End PreRoutingRule.\n\n");
        return true;
    }
    public boolean postRoutingRule(CaseType caseType,
                                   CalloutMediatorMessage calloutMediatorMessage,
                                   CalloutMediatorMessage calloutMediatorMessage1,
                                   Throwable throwable) {
        System.out.println("Start PostRoutingRule.");
        String req_sPayload = "null";
        String req_sPayload_org = "null";
        String rep_sPayload = "null";
        String rep_sPayload_org = "null";
        for (Iterator msgIt =
            calloutMediatorMessage.getPayload().entrySet().iterator();
            msgIt.hasNext(); ) {
            Map.Entry msgEntry = (Map.Entry)msgIt.next();
            Object msgKey = msgEntry.getKey();
            Object msgValue = msgEntry.getValue();
            if(msgKey.equals("request"))
                req_sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
        }
        req_sPayload_org = req_sPayload;
        String tobeReplaced = "PRE_ROUTING_RULE";
        String replaceWith = "PRE_ROUTING_RULE_|_POST_ROUTING_RULE_REQUEST";
        int start = req_sPayload.indexOf(tobeReplaced);
        StringBuffer sb = new StringBuffer();
        sb.append(req_sPayload.substring(0, start));
        sb.append(replaceWith);
        sb.append(req_sPayload.substring(start + tobeReplaced.length()));
        String changedPayload = sb.toString();
        XMLDocument changedoc;
        try {
            changedoc = XmlUtils.getXmlDocument(changedPayload);
            String mykey = "request";
            calloutMediatorMessage.addPayload(mykey, changedoc);
            // calloutMediatorMessage.getPayload().put(mykey, changedoc);
        } catch (Exception e) {
        }
        for (Iterator msgIt =
            calloutMediatorMessage1.getPayload().entrySet().iterator();
            msgIt.hasNext(); ) {
            Map.Entry msgEntry = (Map.Entry)msgIt.next();
            Object msgKey = msgEntry.getKey();
            Object msgValue = msgEntry.getValue();
            if(msgKey.equals("reply"))
                rep_sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
        }
        rep_sPayload_org = rep_sPayload;
        tobeReplaced = "PRE_ROUTING_RULE";
        replaceWith = "PRE_ROUTING_RULE_|_POST_ROUTING_RULE_REQUEST_REPLY";
        start = rep_sPayload.indexOf(tobeReplaced);
        sb = new StringBuffer();
        sb.append(rep_sPayload.substring(0, start));
        sb.append(replaceWith);
        sb.append(rep_sPayload.substring(start + tobeReplaced.length()));
        changedPayload = sb.toString();
        try {
            changedoc = XmlUtils.getXmlDocument(changedPayload);
            String mykey = "reply";
```

```

calloutMediatorMessage1.addPayload(mykey,changedoc.getDocumentElement());
        // calloutMediatorMessage1.getPayload().put(mykey,
changedoc.getDocumentElement());
    } catch (Exception e) {
    }
    System.out.println("Changed from : \n"+req_sPayload_
org+"\nTo\n"+changedPayload);
    System.out.println("End postRoutingRule\n\n");
    return true;
    }
}

```

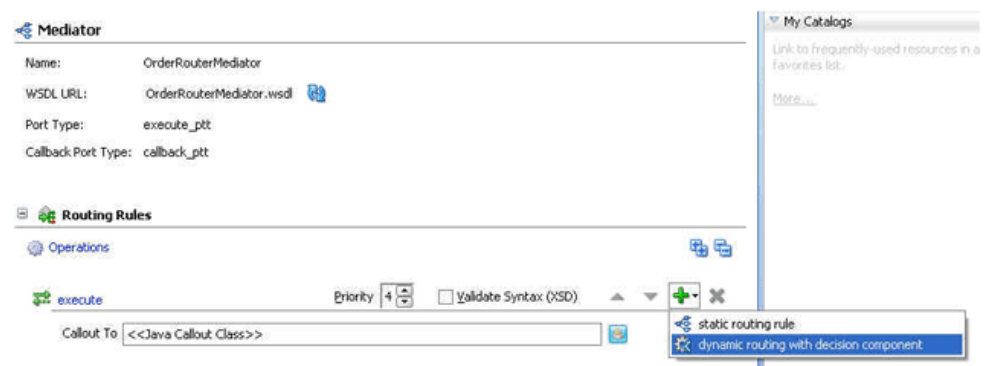
19.2.3 How to Create Dynamic Routing Rules

The basic idea behind dynamic routing is to separate the control logic, which determines the path taken by the process, from the execution of the process. In the dynamic routing scenario, a decision matrix determines the type of Level-2 service to be chosen for each routing. The factors that affect the decision on the type of Level-2 service are channel, customer type, and so on. The solution allows this decision matrix to be modified externally by business analysts without changing the routing. The decision matrix must be evaluated to determine the outbound service.

How to create dynamic routing rules:

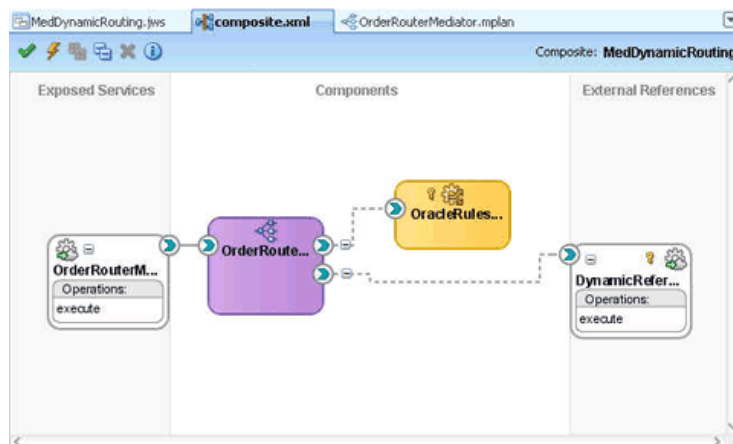
1. Use the dynamic routing rule option of the Mediator Editor, as shown in [Figure 19–32](#):

Figure 19–32 Mediator Editor Displaying Dynamic Routing Rule Option



This creates a new business rule service component that is wired to the Oracle Mediator service component within the SOA composite of the Oracle Mediator service component. The wire links between the business rule service component and the Oracle Mediator service component are considered implementation details and are shown as dotted lines in the SOA Composite Editor, as shown in [Figure 19–33](#).

Figure 19–33 SOA Composite Editor with Wire Links Between the Business Rule and Oracle Mediator Service Components



The business rule service component includes a rule dictionary. The rule dictionary is a metadata container for the rule engine artifacts, such as fact types, rulesets, rules, decision tables and so on. As part of creating the business rule service component, the rule dictionary is preinitialized with the following data.

- **Fact Type Model**

The fact type model is the data model that can be used for modeling rules. The rule dictionary is populated with a fact type model that corresponds to the input of a phase activity in a BPEL process, and some fixed data model that is required as part of the contract between the Oracle Mediator service component and the business rule service component.

- **Ruleset**

A ruleset is a container of rules used as a kind of grouping mechanism for rules. A ruleset can be exposed as a service. As part of creating the business rule service component, one ruleset is created within the rule dictionary.

- **Decision Table (or matrix)**

From a rule engine perspective, a decision table is a collection of rules with the same fact type model elements in the condition and action part of the rules. The decision table enables you to visualize rules in a tabular format. As part of creating the business rule service component, a new decision table is created within the ruleset.

- **Decision Service**

As part of creating the business rule service component, a decision service is created to expose the ruleset as a service of the business rule service component. The service interface is used by the Oracle Mediator service component to evaluate the decision table.

After all the required artifacts of the phase activity are created, the wizard starts modeling the phase decision matrix (PDM). The wizard launches the Business Rules Designer of Oracle JDeveloper and enables you to edit the phase decision matrix. [Figure 19–34](#) shows a sample decision table within the Business Rules Designer.

Figure 19–34 Sample Decision Table Within the Rule Designer

Conditions	R1	R2	R3	R4
C1 CustomerData.type	otherwise	"GOLD"	"Silver"	"Platinum"
Actions				
A1 assert new DynamicRo...	☐	☐	☐	☐
caseName:String	"dynamic_case"	"dynamic_case"	"dynamic_case"	"dynamic_case"
cbkOperation:String	"callback"	"callback"	"callback"	"callback"
executionType:String	"direct"	"direct"	"direct"	"direct"
onCbOperation:String	"callback"	"callback"	"callback"	"callback"
serviceBindingInfo:String	"http://localhost:8001/defaultService"	"http://localhost:8001/GoldService"	"http://localhost:8001/SilverService"	"http://localhost:8001/PlatinumService"
serviceOperations:String	"execute"	"execute"	"execute"	"execute"
serviceReference:String	"DynamicReference_OrderRouterMediator_execute"	"DynamicReference_OrderRouterMediator_execute"	"DynamicReference_OrderRouterMediator_execute"	"DynamicReference_OrderRouterMediator_execute"

- Once the dynamic routing is created, you can modify the associated decision matrix by clicking **Edit Dynamic Rules**. This launches the Business Rules Designer and enables modification of the associated decision table of the business rule service component. After you create dynamic routing for the Oracle Mediator service component, you cannot return to static routing without deleting the dynamic routing. Currently, there is no option for mixing these two types of routing.

The Mediator Editor looks as shown in [Figure 19–35](#) after the dynamic routing option is chosen.

Figure 19–35 Mediator Editor with a Dynamic Routing Rule

The changes in Source view are as follows.

```
<Mediator name="Shipment" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.oracle.com/sca/1.0/mediator">
  <operation name="execute" deliveryPolicy="AllOrNothing" priority="0">
    <switch decisionServiceRef="Phase1DecisionService"
      decisionServiceOperation="executeFunction"></switch>
  </operation>
</Mediator>
```

The `switch` element contains the decision service reference and operation details to enable the Oracle Mediator service component to invoke the decision service in runtime for obtaining the dynamic routing decisions. Dynamic decisions are returned by the business rule service engine user configuration at runtime.

External service invocation contains an extra attribute called **bindingInfo**, which contains binding information to make the invocation dynamic.

19.2.4 What You May Need to Know About Using Dynamic Routing Rules

Note the following limitations on using dynamic routing rules with Oracle Mediator:

- As of now, only SOAP bindings are supported. There is a dummy SOAP binding in the `composite.xml` file. This endpoint is overridden by Oracle Mediator in runtime through an NM property. Therefore, outbound services can be called only over SOAP.
- Payload manipulation is limited for dynamic routing rules. No assignment, transformation, or validation can be performed.
- The reference WSDL file (layer 2 or called references) should have the same abstract WSDL file as the phase reference that gets automatically created.
- Dynamic routing is not possible for Oracle Mediators with a synchronous or one-way interface.

19.2.5 How to Define Default Routing Rules

Oracle Mediator processes messages depending on the conditions specified in the routing rules. In some cases, an Oracle Mediator may not process an incoming message because the message does not satisfy any of the conditions specified in the routing rules. You can define a default routing rule for such messages. The default routing rule is executed when none of the conditions of other routing rules are satisfied.

A default routing rule is the same as the routing rules discussed in [Section 19.2.2, "How to Create Static Routing Rules."](#) The only difference between a default routing rule and other routing rules is that a default routing rule does not have any condition associated with it. Otherwise, a default routing rule is the same as other routing rules in every other aspect, such as target service, response handling, fault handling, and so on.

Notes:

- Default rules are available only for static routing rules.
 - You cannot specify a default routing rule for an Oracle Mediator service component with dynamic routing rules because you cannot define both static and dynamic routing rules in the same Oracle Mediator service component.
-
-

19.2.5.1 Default Rule Scenarios

A default routing rule can be either a sequential rule or a parallel rule. A default routing rule, whether sequential or parallel, is guaranteed to be executed when no other routing rule condition is satisfied. When the default rule is executed, the Oracle Mediator audit trail shows that the filter conditions of all the routing rules failed, and the filter condition of the default routing rule passed and was executed.

[Example 19-21](#) provides details.

Example 19-21 Default Rule Scenarios

ActivityJan 7, 2010 4:35:15 PM
Message onCase "fileout2.Write"

Jan 7, 2010 4:35:15 PM

Message Evaluation of xpath condition " No Filter (DEFAULT CASE) " resulted true

You can define all routing rules, including default routing rules, as either sequential or parallel routing rules, so the expected behavior of routing rules varies. The following sections discuss each combination and the expected behavior:

Sequential Default Routing Rule

You can have the following possible scenarios with a sequential default routing rule:

- **All the other routing rules of the Oracle Mediator are sequential:** This is the simplest case in which all the routing rules, including the default routing rule, are of a sequential type. Runtime evaluates the filter conditions of all routing rules and, if none of the filter conditions are matched, then the default sequential routing rule is executed. Default sequential routing rule execution happens in the same transaction as the incoming message. After the default rule is executed, a post Java callout occurs.
- **At Least One of the Routing Rules of the Oracle Mediator are parallel:** This is a complex case in which the default routing rule is sequential and at least one of the other routing rules is parallel. The default behavior at runtime is to execute all sequential routing rules first and then execute parallel routing rules. Therefore, this is a tricky situation because a default rule should be executed only after all other routing rules are evaluated to be false.

In this case, the server first evaluates the filter condition of parallel rules before evaluating the default routing rule filter condition. If none of the other filter conditions are matched, then the default sequential routing rule is executed.

Parallel Default Routing Rule

You can have the following possible scenarios with a parallel default routing rule:

- **All the other routing rules of the Oracle Mediator are parallel:** This is a straightforward case. The default routing rule is not executed if any of the filter conditions specified in the other routing rules are matched. If none of the filter conditions are matched, then the default routing rule is executed asynchronously.
- **Other Routing Rules of the Oracle Mediator are sequential or parallel:** This is a complex but common use case in which there are other sequential or parallel routing rules available, and the default routing rule is parallel. The default routing rule is not executed if any of the other sequential or parallel routing rule criteria is matched. If none of the conditions are matched, then the default routing rule is executed asynchronously.

Note: The fact that the default routing rule is executed automatically implies that the default routing rule is the only case that was executed for the given Oracle Mediator service component. Similarly, if an Oracle Mediator service component has one routing rule without any filter condition and also has a default routing rule, then the default routing rule is never executed.

19.2.5.2 Default Rule Target

The target of the default routing rule is the same as the supported targets of any other existing routing rule. This indicates that the target can be a service, an event, or an echo. Similarly, the response from the default routing rule target service can be

forwarded or returned to the original caller. If the target service returns a fault, then the fault is handled in the same way as it is handled in any other routing rule.

Note: If exceptions occur while evaluating or executing other routing rules, then the default routing rule is not executed.

19.2.5.3 Default Rule: Validation, Transformation, and Assign Functionality

Schematron validation, transformation, and assign functionality for the default routing rule works in the same way as other routing rules.

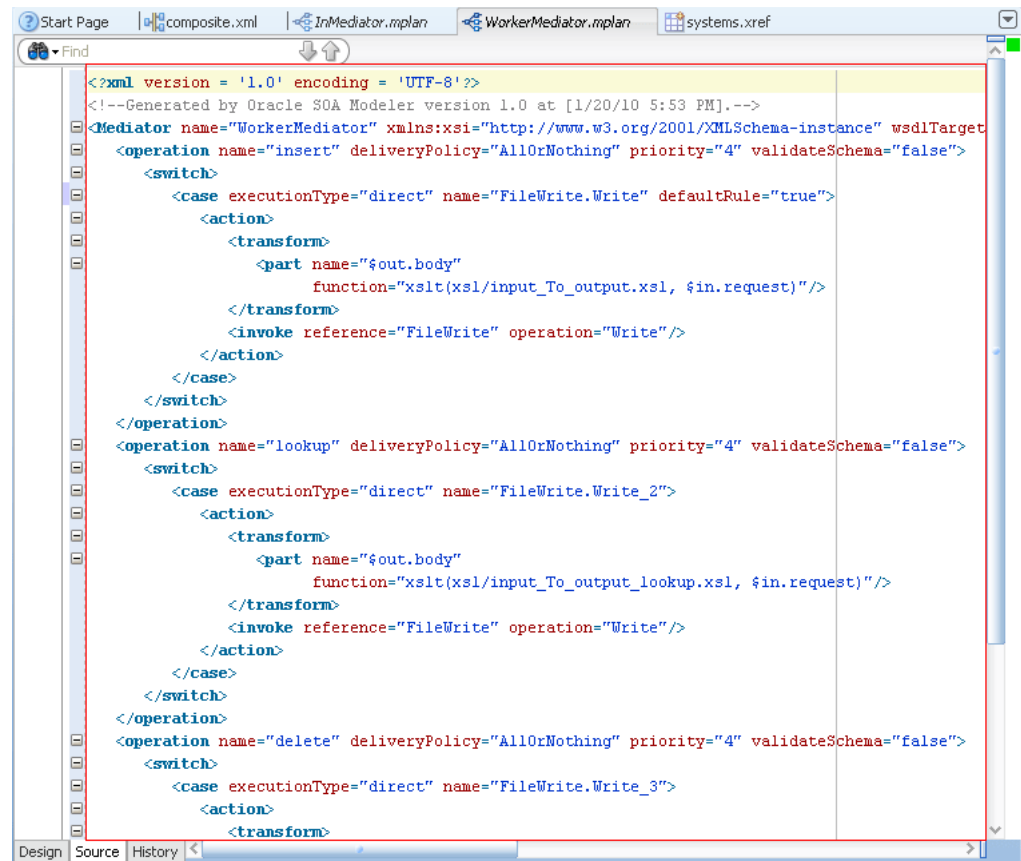
19.2.5.4 Default Rule: Java Callouts

The current behavior of a pre-Java callout or post-Java callout works in the same way as for other routing rules. For the purpose of Java callouts, the default routing rule is considered another routing rule. Therefore, for the scenarios in which the default routing rule is executed, the `postRouting()` callback method occurs only after the default routing rule is executed.

Note: The post-Java callouts occur after the execution of sequential rules and do not wait for the parallel rules to complete execution. Therefore, if the default routing rule is sequential, then the `postRouting()` callback method occurs after executing the default routing rule. If the default routing rule is parallel, then the `postRouting()` callback occurs after all sequential rules are executed and does not wait for the execution of the parallel default routing rule.

19.2.5.5 Default Rule: Oracle Mediator .mplan File

To set a routing rule as the default one, click the **Set as Default Routing Rule** icon shown on [Figure 19-2](#). The `.mplan` file changes, as shown in [Figure 19-36](#).

Figure 19–36 .mplan File of an Oracle Mediator with a Default Routing Rule

19.3 Creating an Oracle Mediator for Routing Messages

The CustomerRouter use case provides an overview of how to use an Oracle Mediator in a SOA composite sample application to route messages. For downloading the sample files mentioned in this section, visit the following URL:

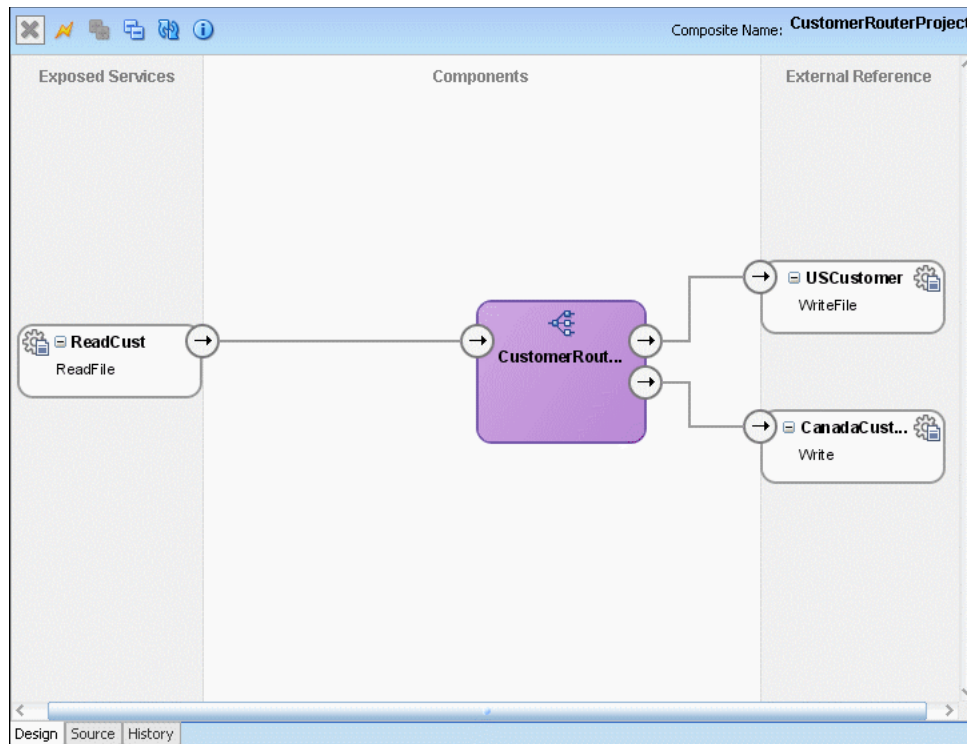
<https://soasamples.samplecode.oracle.com>

The files are provided in the Basic Routing Sample for Oracle Mediator.

The CustomerRouter use case consists of the following steps:

1. Legacy customer files are retrieved from a directory by an adapter service named ReadCust.
2. The ReadCust adapter service sends the file data to the CustomerRouter Oracle Mediator.
3. The CustomerRouter Oracle 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 Oracle Mediator then transforms the message to the structure required by the adapter reference.
5. The external reference delivers the message to its associated external application.

Figure 19–37 provides an overview of the CustomerRouter use case.

Figure 19–37 Overview of CustomerRouter Use Case

19.3.1 How to Create the CustomerRouter Use Case

This section provides the design-time tasks for creating, building, and deploying the use case.

19.3.1.1 Task 1: How to Create an Oracle JDeveloper Application and a Project

To create an Oracle JDeveloper application and a project:

1. In Oracle JDeveloper, click **File** and select **New**.
The New Gallery dialog appears.
2. In the New Gallery, expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **SOA Application** and click **OK**.
The Create SOA Application wizard appears.
4. In the **Application Name** field, enter `CustomerRouter` and then click **Next**.
The Name your project page appears.
5. In the **Project Name** field, enter `CustomerRouterProject` and click **Next**.
The Configure SOA settings page appears.
6. From the **Composite Template** list, select **Empty Composite** and then click **Finish**.
The Application Navigator of Oracle JDeveloper is populated with the new application and the project, and the SOA Composite Editor contains a blank palette.
7. From the **File** menu, select **Save All**.

19.3.1.2 Task 2: How to Create the CustomerRouter Oracle Mediator Service Component

To create the CustomerRouter Oracle Mediator service component:

1. From the Component Palette, select **SOA**.
2. Drag and drop a **Mediator** icon in the **Components** section.
The Create Mediator dialog is displayed.
3. In the **Name** field, enter `CustomerRouter`.
4. From the **Templates** list, select **Define Interface Later**.
5. Click **OK**.

A Oracle Mediator with name **CustomerRouter** is created.

19.3.1.3 Task 3: How to Create a File Adapter Service

You must create a file adapter service named `ReadCust` to read the XML files from a directory.

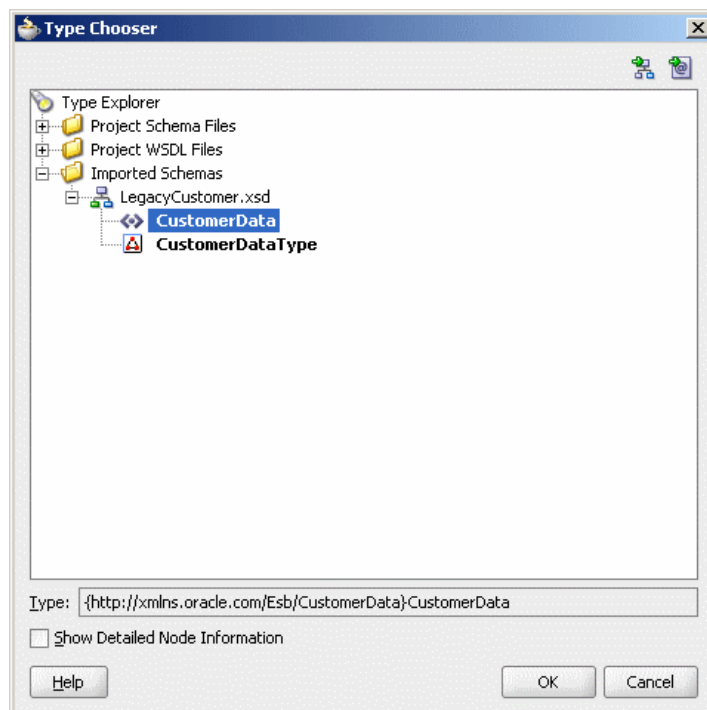
Note: Oracle Mediator may process the same file twice when run against Oracle Real Application Clusters (Oracle RAC) planned outages. This is because a file adapter is a non-XA compliant adapter. Therefore, when it participates in a global transaction, it may not follow the XA interface specification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

To create a file adapter service:

1. From the Component Palette, select **SOA**.
2. Select **File Adapter** and drag it to the **Exposed Services** swimlane.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter `ReadCust`.
5. Click **Next**.
The Adapter Interface page is displayed.
6. Select **Define from operation and schema (specified later)** and click **Next**.
The Operation page is displayed.
7. In the **Operation Type** field, select **Read File**.
8. In the **Operation Name** field, replace **Read** with `ReadFile`.
9. Click **Next**.
The File Directories page is displayed.
10. In the **Directory for Incoming Files (physical path)** field, enter the directory from which you want to read the files. For example, enter `C:\Customer\In`.
11. Click **Next**.
The File Filtering page is displayed.

12. In the **Include Files with Name Pattern** field, enter *.xml, and then click **Next**.
The File Polling page is displayed.
13. Change the **Polling Frequency** field value to **10 seconds**, and then click **Next**.
The Messages page is displayed.
14. To the right of the **URL** field, click **Search**.
The Type Chooser dialog is displayed.
15. Click **Import Schema File**.
The Import Schema File dialog is displayed.
16. To the right of the **URL** field, click **Search** and select the **LegacyCustomer.xsd** file present in the **Samples** folder.
17. Click **OK**.
18. Expand the navigation tree to **Type Explorer\Imported Schemas\LegacyCustomer.xsd** and select **CustomerData**, as shown in [Figure 19–38](#).

Figure 19–38 Type Chooser - CustomerData



19. Click **OK**.
The Adapter Configuration wizard appears, as shown in [Figure 19–39](#).

Figure 19–39 Adapter Configuration Wizard – Messages page

20. Click **Next**.

The Finish page is displayed.

21. Click **Finish**.

22. From the **File** menu, select **Save All**.

19.3.1.4 Task 4: How to Create a File Adapter Reference

You must create a USCustomer file adapter reference.

To create a file adapter reference:

1. From the Component Palette, select **SOA**.
2. Select **File Adapter** and drag it to the **External References** swimlane.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter **USCustomer**.
5. Click **Next**.
The Adapter Interface page is displayed.
6. Select **Define from operation and schema (specified later)** and click **Next**.
The Operation page is displayed.
7. Click **Next**.
The Operation page is displayed.
8. In the **Operation Type** field, select **Write File**.

9. In the **Operation Name** field, enter `WriteFile`.
10. Click **Next**.

The File Configuration page is displayed.
11. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory in which you want to write the files.

For example, `C:\Customer\out`.
12. In the **File Naming Convention** field, enter `customer_%SEQ%.xml` and click **Next**.

The Messages page is displayed.
13. To the right of the **URL** field, click **Search**.

The Type Chooser dialog is displayed.
14. Click **Import Schema File**.

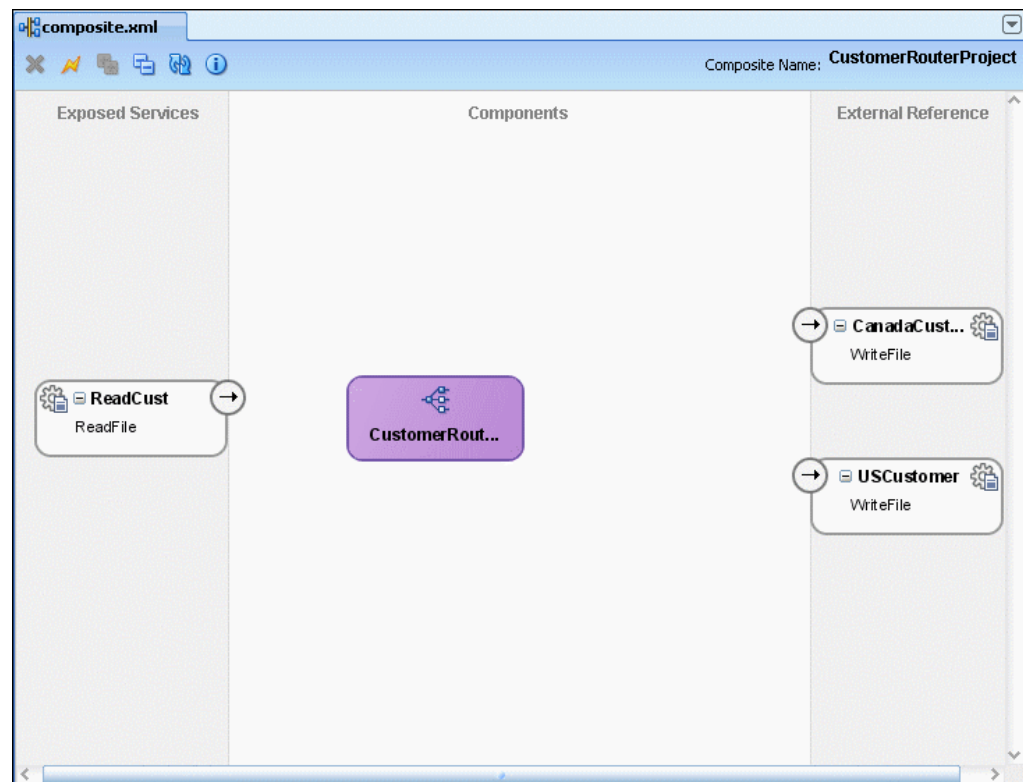
The Import Schema File dialog is displayed.
15. To the right of the **URL** field, click **Search** and select the **USCustomer.xsd** file present in the **Samples** folder.
16. Click **OK**.
17. Expand the navigation tree to **Type Explorer\Imported Schemas\USCustomer.xsd**, and then select **Customer**.
18. Click **OK**.
19. Click **Next**.

The Finish page is displayed.
20. Click **Finish**.
21. From the **File** menu, click **Save All**.

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

[Figure 19-40](#) shows how the SOA Composite Editor appears after performing this task.

Figure 19–40 Oracle Mediator Service Component with Adapter Services and References



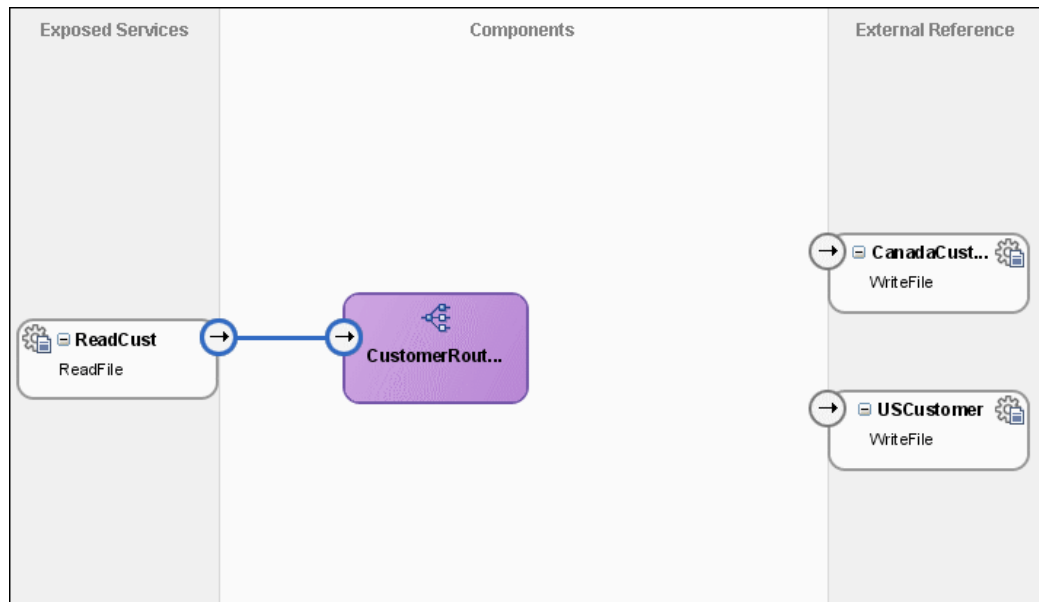
19.3.1.5 Task 5: How to Specify Routing Rules

You must specify the path that messages take from the ReadCust adapter service to external references.

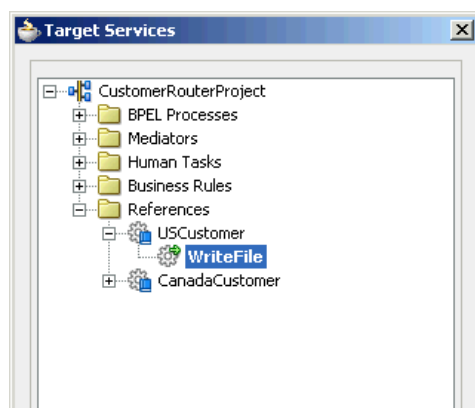
To specify routing rules:

1. Connect the **ReadCust** service to the **CustomerRouter** Oracle Mediator, as shown in [Figure 19–41](#).

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

Figure 19–41 Connecting the ReadCust Service to the CustomerRouter Oracle Mediator

2. Double-click the **CustomerRouter** Oracle Mediator in the Mediator Editor.
3. In the **Routing Rules** section, click **Add** to the extreme right side of **ReadFile**, and then click **static routing rule**.
The Target Type dialog is displayed.
4. Click **Service**.
The Target Services dialog is displayed.
5. Navigate to **CustomerRouterProject > References > USCustomer** and select **WriteFile**, as shown in [Figure 19–42](#).

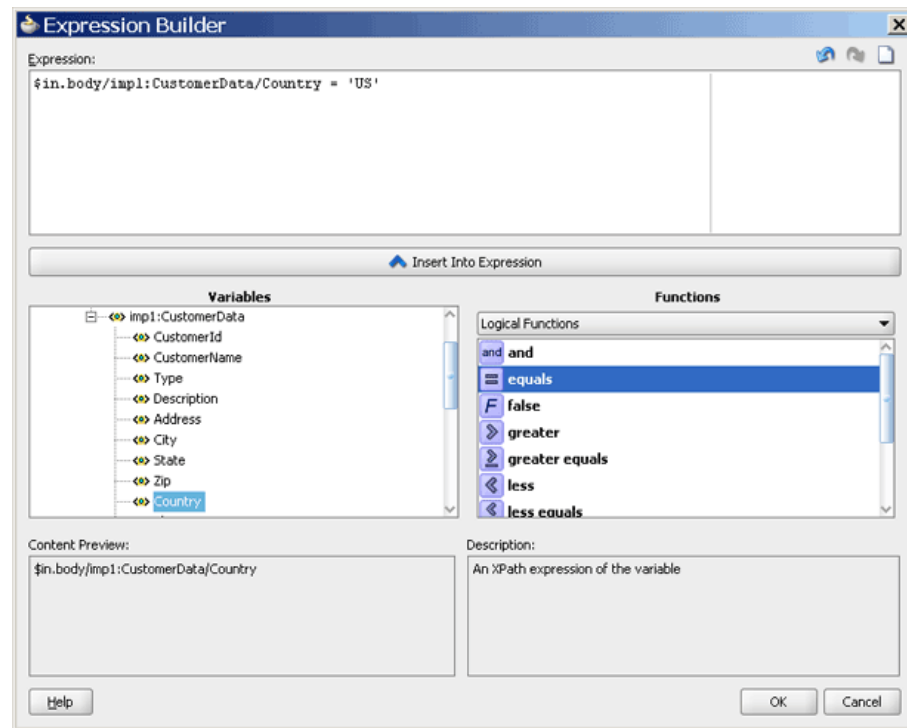
Figure 19–42 Target Services Dialog

6. Click **OK**.
The **Routing Rules** section is displayed.
7. Next to the **<<Filter Expression>>** field, click the **filter** icon to create a filter expression for this routing rule.
The Expression Builder dialog is displayed.

8. In the **Variables** field, navigate to **Variables> in > body > imp1:CustomerData**, and then select **Country**.
9. Double-click **Country**.

The **Country** node is added in the **Expression** field, as shown in Figure 19–43.

Figure 19–43 Expression Builder Dialog



10. Modify the expression as follows:

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

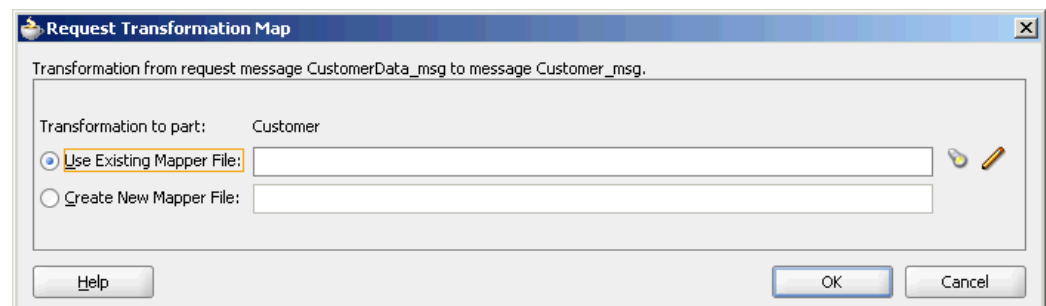
11. Click **OK**.

The **<<Filter Expression>>** field of the **Routing Rules** section is populated with the expression.

12. To the right of the **Transform Using** field, click the icon.

The Request Transformation Map dialog is displayed, as shown in Figure 19–44.

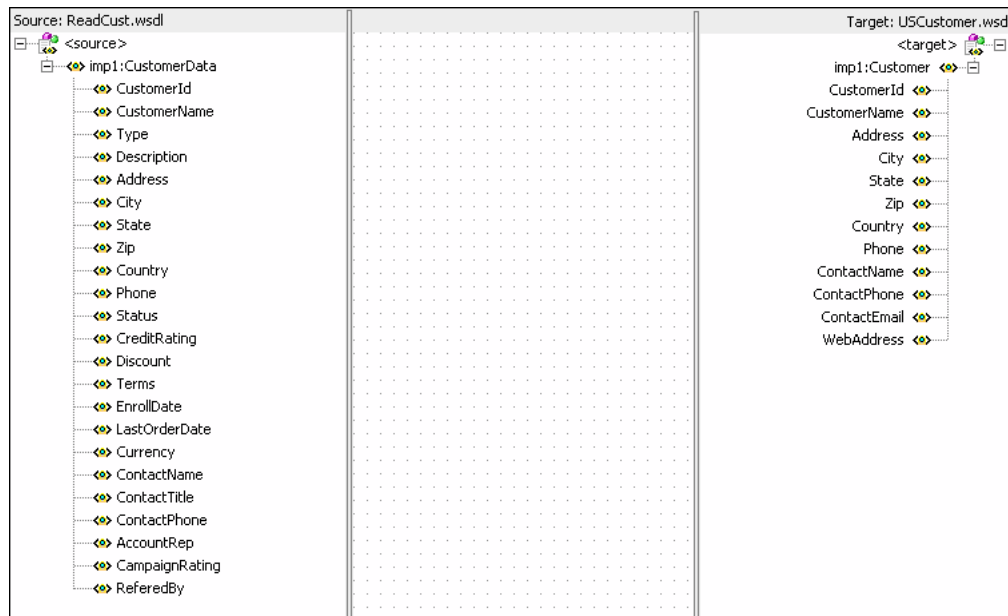
Figure 19–44 Request Transformation Map



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

A **CustomerData_To_Customer.xsl** file is added, as shown in [Figure 19-45](#).

Figure 19-45 *CustomerData_To_Customer.xsl File – Initially*



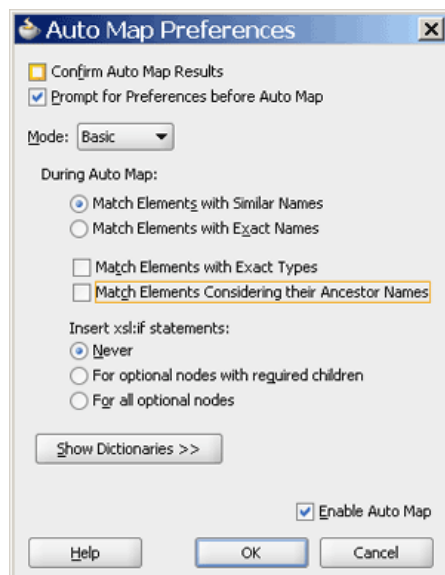
14. Drag and drop the **imp1:CustomerData** source element to the **imp1:Customer** target element.

The Auto Map Preferences dialog is displayed.

15. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names**.

The Auto Map Preferences dialog is shown in [Figure 19-46](#).

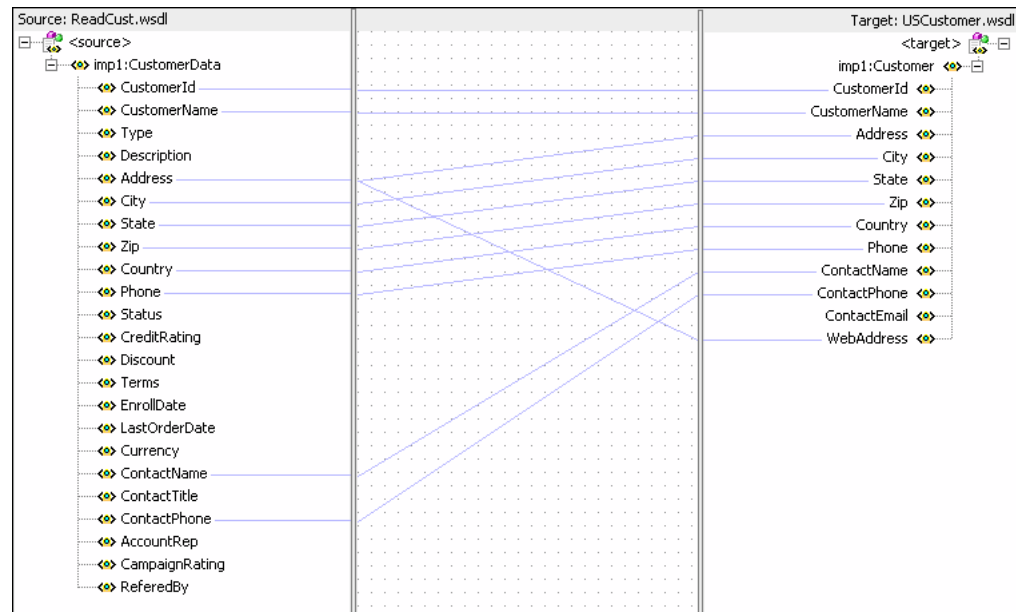
Figure 19-46 *Auto Map Preferences Dialog*



16. Click OK.

The **CustomerData_To_Customer.xsl** file appears, as shown in [Figure 19-47](#).

Figure 19-47 CustomerData_To_Customer.xsl Tab – Auto Mapped Connections



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

18. Repeat the procedures mentioned in Step 3 through Step 17 to create a **CanadaCustomer** reference as the target service. In the Expression Builder dialog, specify the following expression:

Note: For repeating the steps, you must re-enter the Mediator Editor by closing it or by clicking the **CustomerRouter.mplan** tab above the editor.

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

After performing all the steps described in this section, the SOA Composite Editor appears, as shown in [Figure 19-37](#).

19.3.1.6 Task 6: How to Create an Application Server Connection

An application server connection is required for deploying your SOA composite application. For information about creating an application server connection, see [Section 40.7.1.1.1, "Creating an Application Server Connection."](#)

19.3.1.7 Task 7: How to Deploy the CustomerRouterProject

Deploying the **CustomerRouterProject** composite application to an application server consists of following steps:

- Creating an application deployment profile
- Deploying the application deployment profile to an application server

For detailed information about these steps, see [Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper."](#)

19.3.2 Running and Monitoring the CustomerRouterProject Application

After deploying the CustomerRouterProject application, you can run it by copying the input XML files to the input folder. The payload files are written to the specified output directories.

For monitoring the running instance, you can use Oracle Enterprise Manager Fusion Middleware Control at the following URL:

`http://hostname:port_number/em`

where *hostname* is the host on which you installed the Oracle SOA Suite infrastructure and *port_number* is the port of the server on which Oracle Enterprise Manager Fusion Middleware Control is installed.

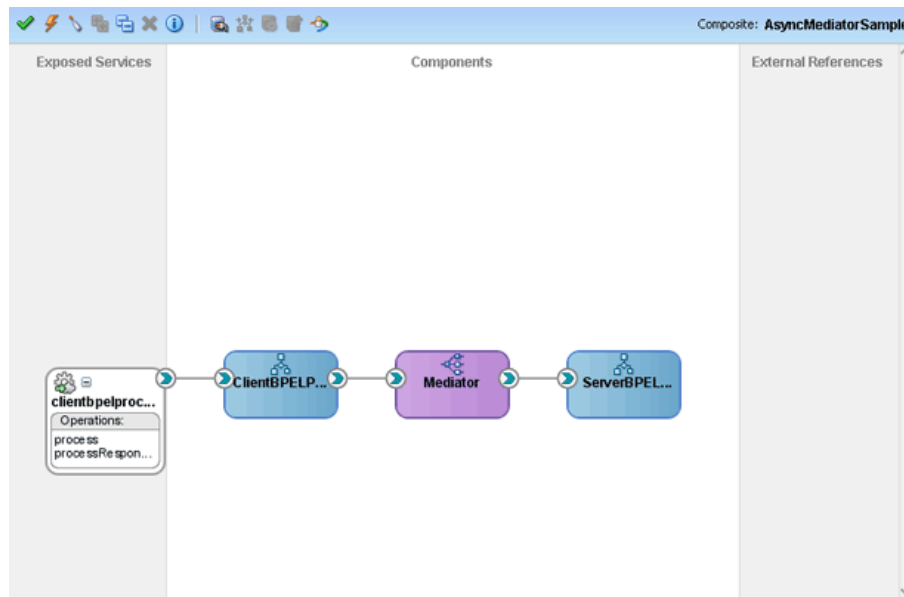
For detailed information about these steps, see [Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper."](#)

19.4 Creating an Asynchronous Request and Response Using Oracle Mediator

This sample demonstrates an asynchronous request response scenario using Oracle Mediator. This composite has a client BPEL process invoking an Oracle Mediator, which invokes a server BPEL process. All the invocations are done as an asynchronous request response.

[Figure 19–48](#) provides an overview of the AsyncMediator use case.

Figure 19–48 Overview of AsyncMediator Use Case



For downloading the sample files mentioned in this section, visit the following URL:

<https://soasamples.samplecode.oracle.com>

19.4.1 How to Create the AsyncMediator Use Case

This section provides the design-time tasks for creating, building, and deploying the use case. These tasks should be performed in the order in which they are presented.

19.4.1.1 Task 1: How to Create an Oracle JDeveloper Application and Project

To create an Oracle JDeveloper application and a project:

1. In Oracle JDeveloper, click **File** and select **New**.
The New Gallery dialog appears.
2. In the New Gallery, expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **SOA Application** and click **OK**.
The Create SOA Application wizard appears.
4. In the **Application Name** field, enter `AsyncMediator` and then click **Next**.
The Name your project page appears.
5. In the **Project Name** field, enter `AsyncMediatorSample` and click **Next**.
The Configure SOA settings page appears.
6. In the **Composite Template** list, select **Empty Composite** and then click **Finish**.
The Application Navigator of Oracle JDeveloper is populated with the new application and the project, and the SOA Composite Editor contains a blank palette.
7. From the **File** menu, click **Save All**.

19.4.1.2 Task 2: How to Create a Server BPEL Process

To create a server BPEL process:

1. In the Application Navigator, double-click **composite.xml**. The SOA Composite Editor is displayed.
2. From the Component Palette, select **SOA**.
3. Drag and drop a **BPEL Process** into the **Components** section.
The Create BPEL Process dialog is displayed.
4. In the **Name** field, enter `ServerBPELProcess`.
5. From the **Template** list, select **Asynchronous BPEL Process**.
6. Deselect **Expose as a SOAP service** and click **OK**. The **ServerBPELProcess** is created in the SOA Composite Editor.

19.4.1.3 Task 3: How to Create an Oracle Mediator Service Component

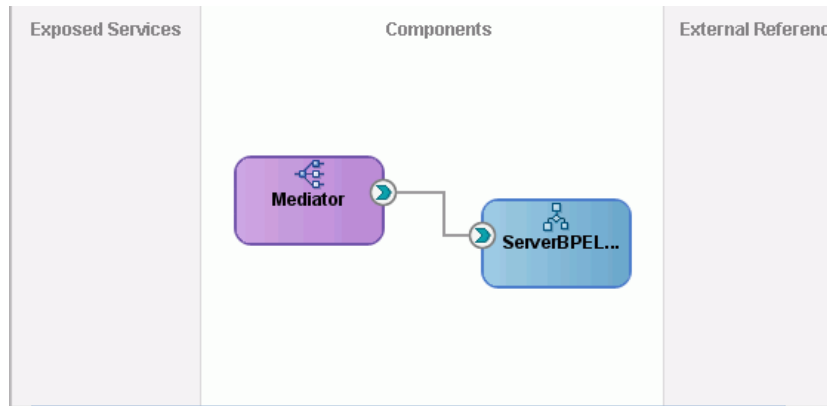
To create an Oracle Mediator service component named Mediator:

1. From the Component Palette, select **SOA**.
2. Drag and drop a **Mediator** into the **Components** section.
The Create Mediator dialog is displayed.
3. In the **Name** field, enter `Mediator`.

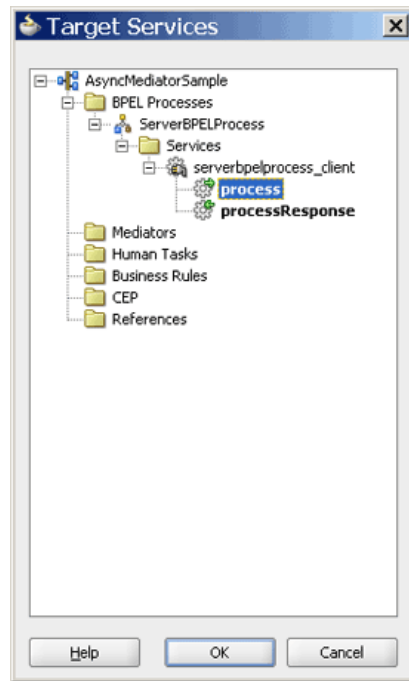
4. From the **Template** list, select **Asynchronous Interface**.
5. Deselect **Create Composite Service with SOAP Bindings**.
6. Click **OK**.

An Oracle Mediator with name **Mediator** is created, as shown in [Figure 19–49](#).

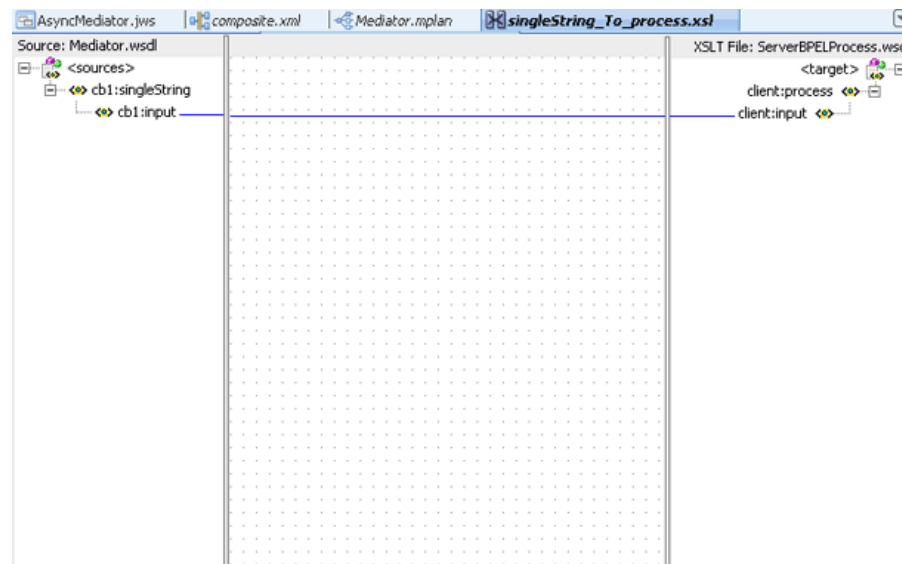
Figure 19–49 Oracle Mediator and ServerBPELProcess in the Composite Window



7. Double-click the **Mediator** Oracle Mediator.
The Mediator Editor is displayed.
8. In the **Routing Rules** section, click **Add** to the far right side of **execute** and then select **static routing rule**.
The Target Type dialog is displayed.
9. Select **Service**.
The Target Services dialog is displayed.
10. Navigate to **AsyncMediatorSample > BPEL Processes > ServerBPELProcess > Services > serverbpelprocess_client > process**, as shown in [Figure 19–50](#).

Figure 19–50 Target Services Dialog

11. Click **OK**.
12. Below the **<<Filter Expression>>** field, click the icon to the right of the **Transform Using** field.
The Request Transformation Map dialog is displayed.
13. Select **Create New Mapper File** and click **OK**.
The XSLT Mapper is displayed and a target file named **singleString_To_process.xml** is added.
14. Drag and drop the **cb1:input** source element to the **client:input** target element.
The Auto Map Preferences dialog is displayed.
15. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names** and click **OK**.
The XSLT Mapper displays, as shown in [Figure 19–51](#).

Figure 19–51 *singleString_To_process.xsl* Window

16. In the **Routing Rules** section, under **Callback**, click the icon to the right of the **Transform Using** field.
The Request Transformation Map dialog is displayed.
17. Select **Create New Mapper File** and click **OK**.
The XSLT Mapper is displayed and a target file named **processResponse_To_singleString.xsl** is added.
18. Drag and drop the **client:processResponse** source element to the **cb1:singleString** target element.
The Auto Map Preferences dialog is displayed.
19. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names** and click **OK**.

19.4.1.4 Task 4: How to Create a Client BPEL Process

To create a client BPEL process:

1. In the Application Navigator, double-click **composite.xml**. The SOA Composite Editor is displayed.
2. From the Component Palette, select **SOA**.
3. Drag and drop a **BPEL Process** to the **Components** section.
The Create BPEL Process dialog is displayed.
4. In the **Name** field, enter **ClientBPELProcess**.
5. From the **Template** list, select **Asynchronous BPEL Process**.
6. Click **OK**.
ClientBPELProcess is created in the SOA Composite Editor.
7. Drag and drop the **ClientBPELProcess** BPEL process to the **Mediator Oracle Mediator**. The SOA Composite Editor appears, as shown in [Figure 19–48](#).

19.4.1.5 Task 5: How to Create the Invoke, Receive, and Assign Activities

To create the invoke activity:

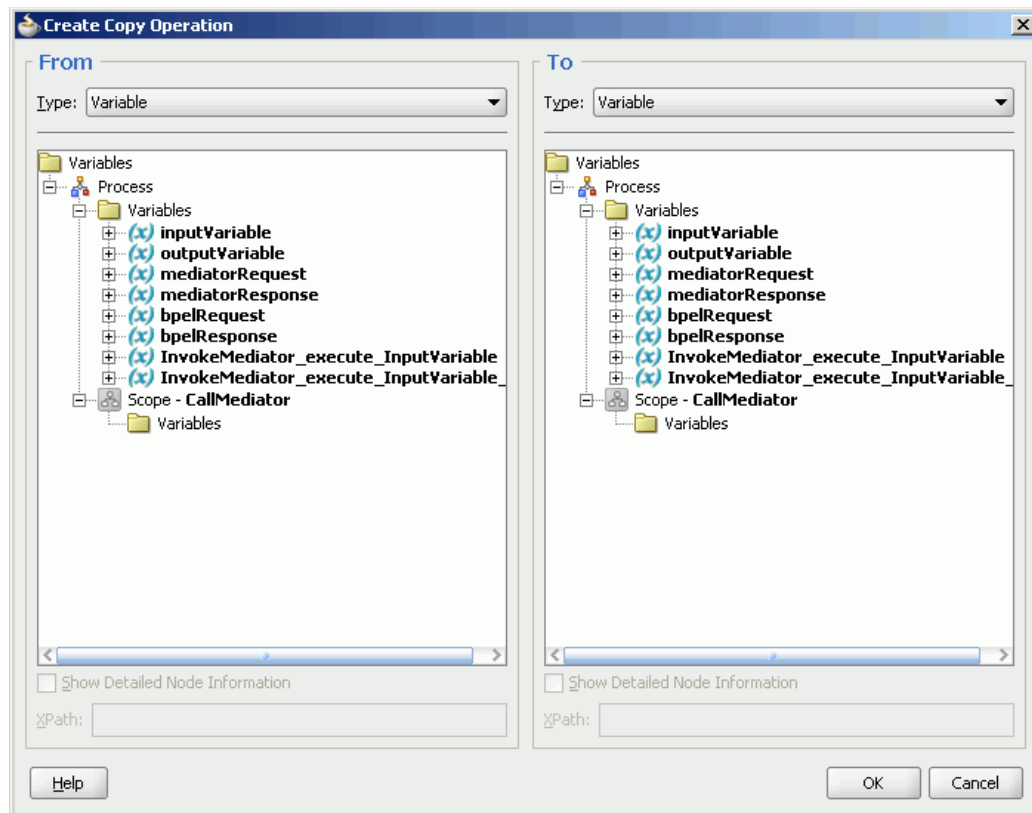
1. Double-click **ClientBPELProcess**. The Oracle BPEL Designer is displayed.
2. Drag and drop an **Invoke** activity from the Component Palette to the design area.
3. Double-click the **Invoke** activity. The Invoke dialog is displayed.
4. In the **Name** field, enter `InvokeMediator`.
5. Next to the **Partner Link** field, click **Browse Partner Links**. The Partner Link Chooser dialog is displayed.
6. Select **Operation - execute**, and click **OK**.
7. To the right of the **Variable** field in the Invoke dialog, click the **Auto-Create Variable** icon. The Create Variable dialog is displayed.
8. In the **Variable** field, enter `InvokeMediator_execute_InputVariable_1` and click **OK**. The Invoke dialog is displayed.
9. Click **OK**. The Oracle BPEL Designer appears.

To create the receive activity:

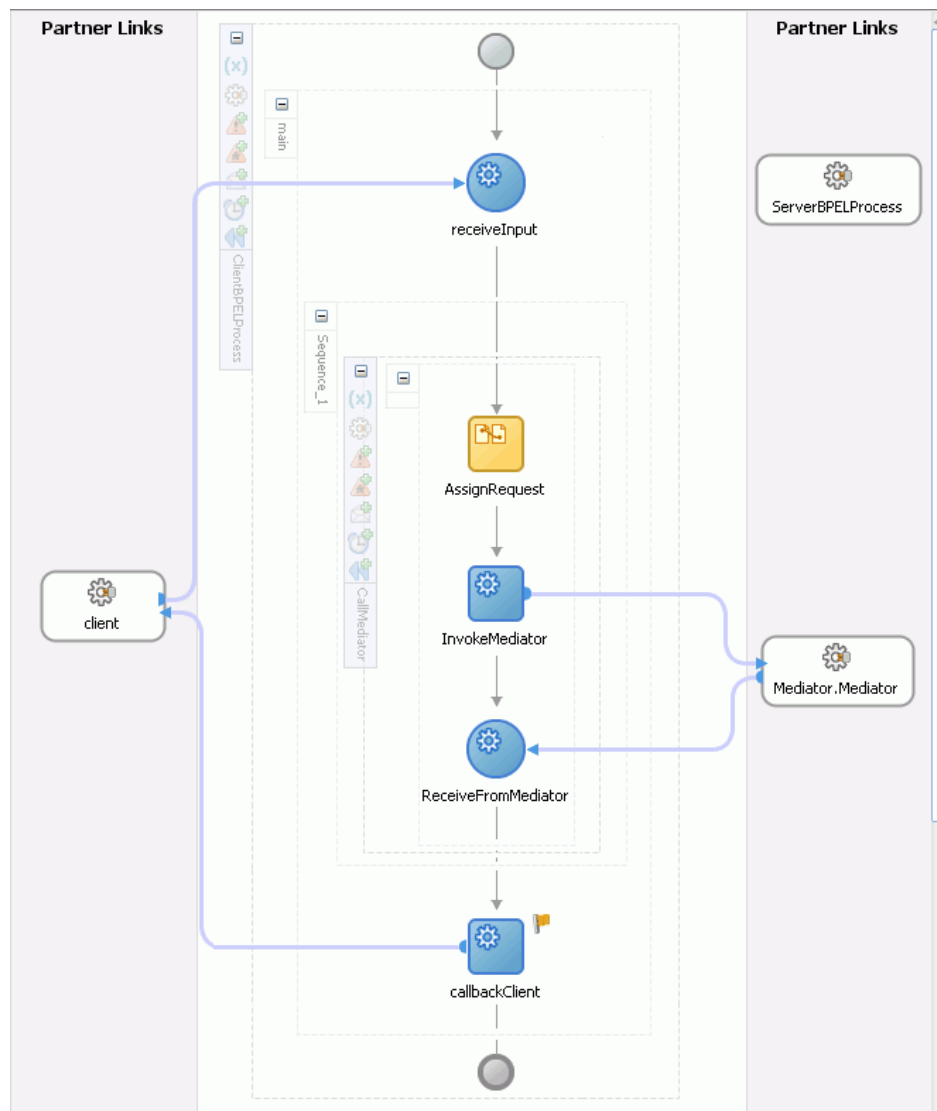
1. Drag and drop a **Receive** activity from the Component Palette to the design area.
2. Double-click the **Receive** activity. The Receive dialog is displayed.
3. In the **Name** field, enter `ReceiveFromMediator`.
4. Next to the **Partner Link** field, click **Browse Partner Links**. The Partner Link Chooser dialog is displayed.
5. Select **Operation - callback**, and click **OK**.
6. To the right of the **Variable** field in the Receive dialog, click the **Auto-Create Variable** icon. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The **Variable** field is populated with the default variable name.
8. Check **Create Instance**, and click **OK**. The Oracle BPEL Designer appears.

To create the assign activity:

1. Drag and drop an **Assign** activity from the Component Palette between the **ReceiveFromMediator** and **InvokeMediator** activities in the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. In the **Name** field, enter `AssignRequest`.
4. Click the **Copy Operation** tab. The Assign dialog is displayed.
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Create the copy operation between the triggers file name and the file variable, as shown in [Figure 19–52](#).

Figure 19–52 *The Create Copy Operation Dialog*

7. Click **OK** in the Create Copy Operation dialog.
8. Click **OK** to return to the Oracle BPEL Designer, as shown in [Figure 19–53](#).

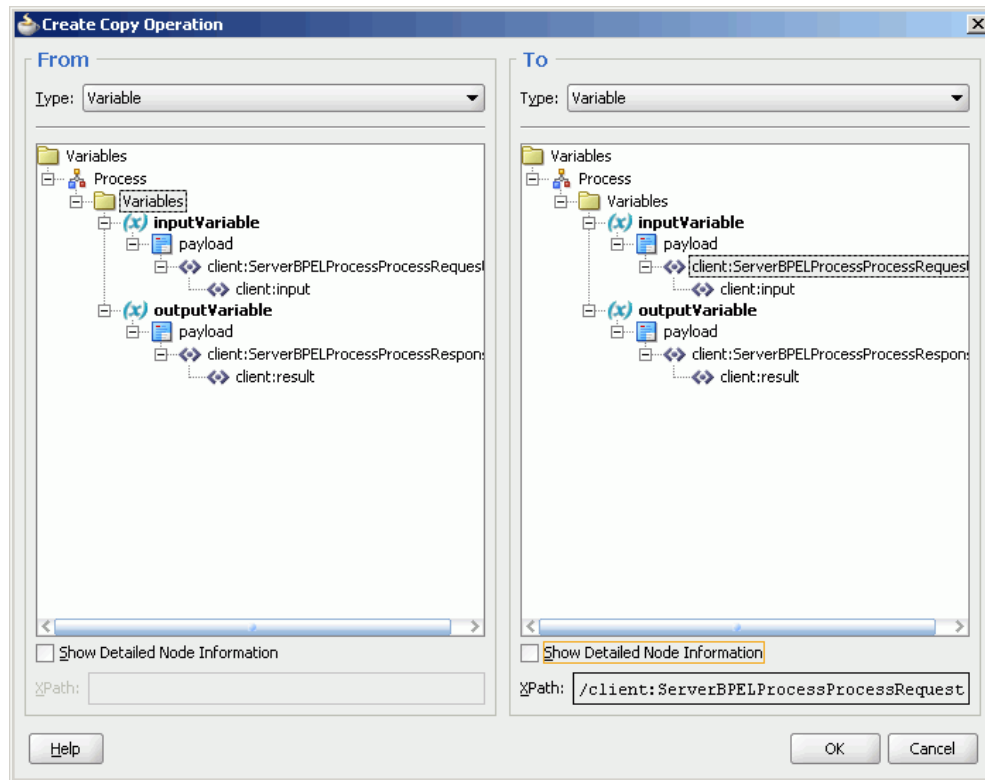
Figure 19–53 The Oracle JDeveloper - ClientBPELProcess.bpel

9. From the **File** menu, select **Save All**.

To create an assign activity in the ServerBPELProcess

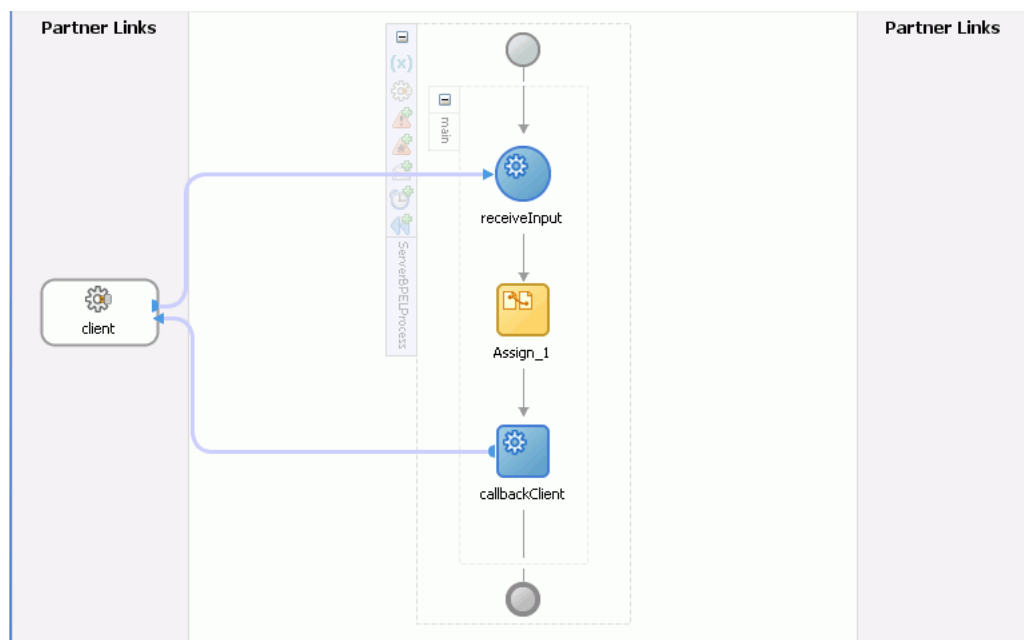
1. Double-click the **ServerBPELProcess** BPEL process. The Oracle BPEL Designer is displayed.
2. Drag and drop an **Assign** activity from the Component Palette between the **receiveInput** and **callbackClient** activities in the design area.
3. Double-click the **Assign** activity. The Assign dialog is displayed.
4. Click the **Copy Operation** tab.
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Create the copy operation between the triggers file name and the file variable, as shown in [Figure 19–54](#).

Figure 19–54 The Create Copy Operation Dialog



7. Click **OK** in the Create Copy Operation dialog.
8. Click **OK** to return to the Oracle BPEL Designer, as shown in [Figure 19–55](#).

Figure 19–55 The Oracle JDeveloper - ServerBPELProcess.bpel



9. From the **File** menu, select **Save All**.

19.4.1.6 Task 6: How to Configure an Application Server Connection

An application server connection is required for deploying your SOA composite application. For information on creating an application server connection, see [Section 40.7.1.1.1, "Creating an Application Server Connection."](#)

19.4.1.7 Task 7: How to Deploy the SOA Composite Application

Deploying the EventMediatorApp composite application to Oracle WebLogic Server consists of following steps:

- Creating an application deployment profile
- Deploying the application to an application server

For detailed information about these steps, see [Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper."](#)

Working with Multiple Part Messages in Oracle Mediator

This chapter describes how to use multiple part (multipart) messages with the Oracle Mediator service component.

This chapter includes the following sections:

- [Section 20.1, "Introduction to Oracle Mediator Multipart Message Support"](#)
- [Section 20.2, "Working with Multipart Request Messages"](#)

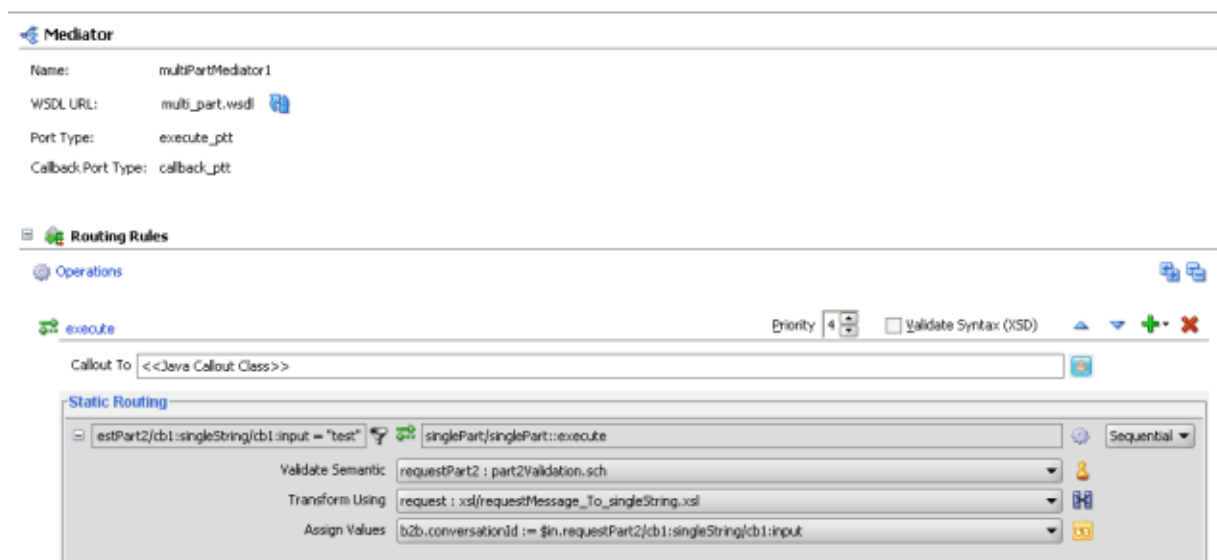
20.1 Introduction to Oracle Mediator Multipart Message Support

Oracle Mediator includes support for importing multipart WSDL files in the Mediator Editor.

Oracle Mediator supports working with multipart source and target messages, which include multipart filter expression building, multipart schema validation, and transformations between multipart source and target messages for requests, replies, faults, and callbacks.

The Mediator Editor with a multipart source appears as shown in [Figure 20–1](#).

Figure 20–1 Mediator Editor for a Multipart Source



20.2 Working with Multipart Request Messages

This section describes how to work with different types of multipart messages.

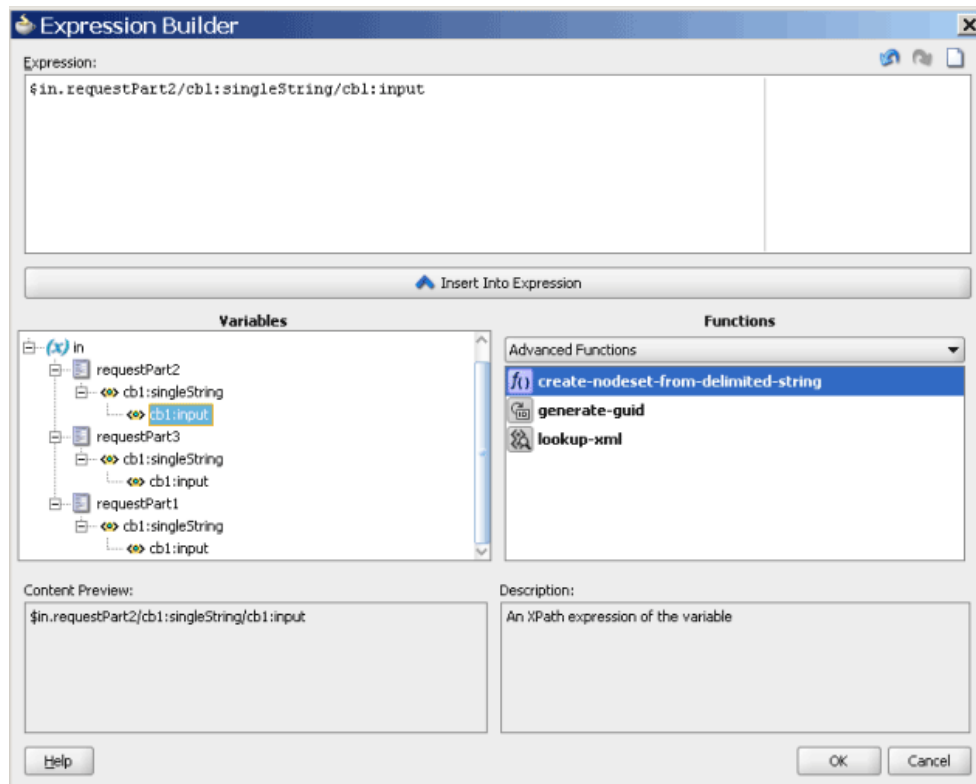
20.2.1 How to Work with Multipart Request Messages

This section describes multipart request messages.

20.2.1.1 How to Specify Filter Expressions

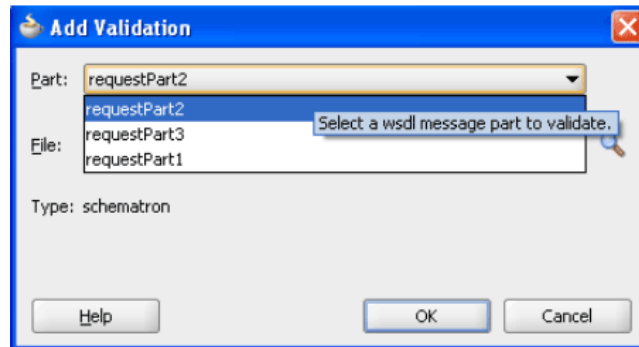
If you specify a filter expression for a multipart message, then the Expression Builder displays all message parts under the **in** variable, as shown in [Figure 20–2](#):

Figure 20–2 Expression Builder for a Multipart Request Source



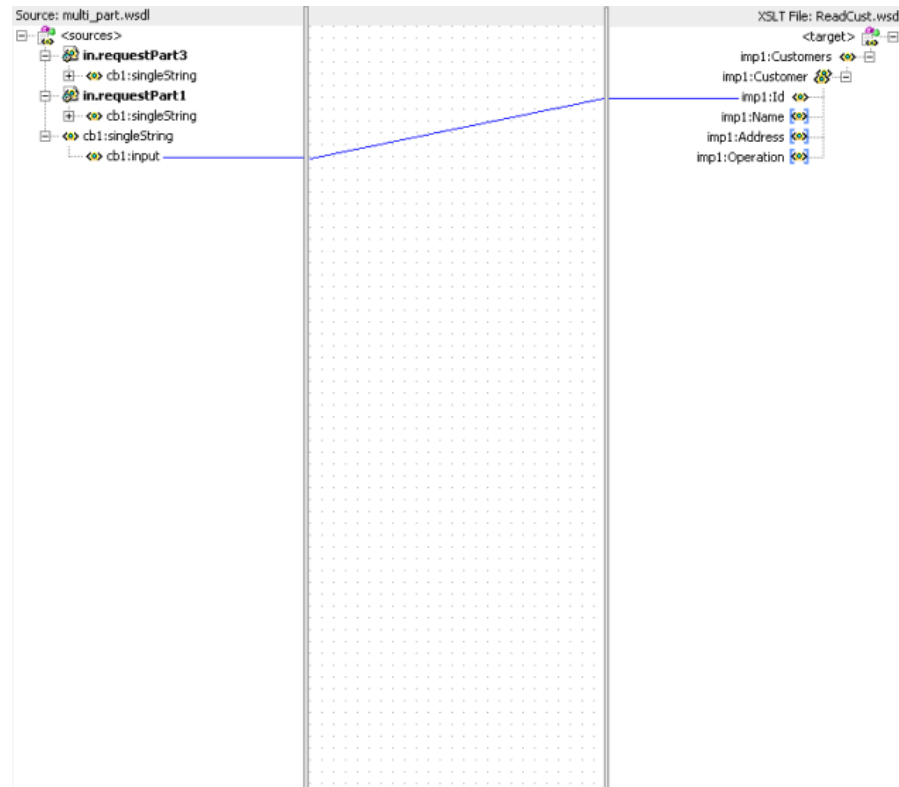
20.2.1.2 How to Add Validations

If you add a validation for a multiple part message, then the Add Validation dialog displays a list of parts from which you can choose one part, as shown in [Figure 20–3](#). You specify a Schematron file for each request message part. Oracle Mediator then processes the Schematron files for the parts.

Figure 20–3 Add Validation Dialog for a Multipart Request Source

20.2.1.3 How to Create Transformations

If you create a new mapper file for a multipart message, then the generated mapper file shows multiple source parts in the XSLT Mapper, as shown in [Figure 20–4](#):

Figure 20–4 XSLT Mapper for a Multipart Request Source

20.2.1.4 How to Assign Values

If you assign values using a source expression, then the Expression Builder shows an **in** variable for each message part. This is the same as specifying filter expressions.

20.2.2 How to Work with Multipart Reply, Fault, and Callback Source Messages

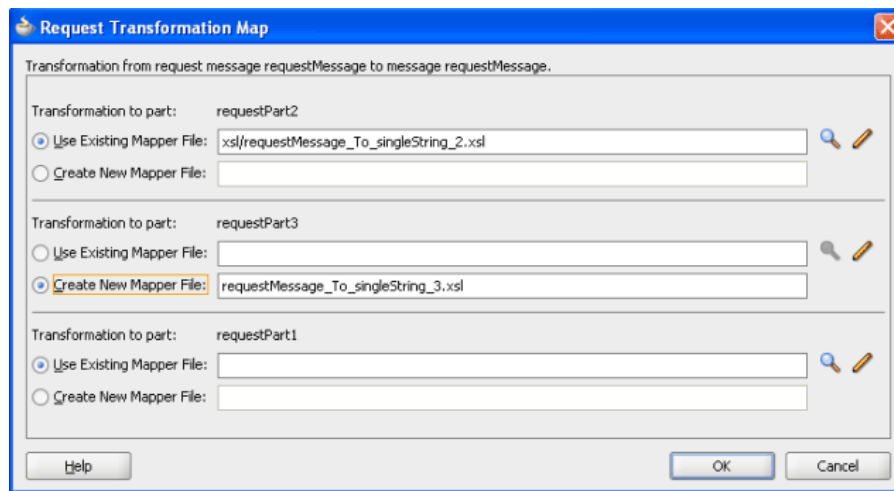
The method to create transformations and assign values to multipart reply, fault, and callback source messages is the same as working with request source messages.

Note: You cannot specify filter expressions or add validations for reply, fault, and callback messages.

20.2.3 How to Work with Multipart Target Messages

If a routing target (that is, a request, reply, fault, or callback) has a multipart message, then the transformation is handled in a slightly different way. This is because the XSLT Mapper does not support multipart targets. In such a case, the Oracle Mediator creates and coordinates a separate mapper file for each target part, as shown in [Figure 20–5](#):

Figure 20–5 Request Transformation Map for a Multipart Routing Target



Using Oracle Mediator Error Handling

This chapter describes how to handle errors with Oracle Mediator.

This chapter includes the following sections:

- [Section 21.1, "Introduction to Oracle Mediator Error Handling"](#)
- [Section 21.2, "Using Error Handling with Oracle Mediator"](#)
- [Section 21.3, "Fault Recovery Using Oracle Enterprise Manager Fusion Middleware Control"](#)
- [Section 21.4, "Error Handling XML Schema Definition Files"](#)

21.1 Introduction to Oracle Mediator Error Handling

Oracle Mediator provides sophisticated error handling capabilities that enable you to configure an Oracle Mediator service component for error occurrences and corresponding corrective actions. Error handling enables an Oracle Mediator to handle errors that occur during the processing of messages and also the exceptions returned by outside web services. You can handle both business faults and system faults with 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 a network not being available. Oracle Mediator provides fault policy-based error handling for system faults.

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

- Fault policies
- Fault bindings
- Error groups

21.1.1 Fault Policies

A fault policy defines error conditions and corresponding actions. Fault policies are defined in the `fault-policies.xml` file. The `fault-policies.xml` file should be created based on the XML schema defined in [Section 21.4.1, "Schema Definition File for fault-policies.xml."](#)

Note: Fault policies are applicable to parallel routing rules only. For sequential routing rules, the fault goes back to the caller. It is the responsibility of the caller to handle the fault. If the caller is an adapter, then you can define rejection handlers on the inbound adapter to take care of the messages that error out (that is, the rejected messages). For more information about rejection handlers, see the *Oracle Fusion Middleware User's Guide for Technology Adapters*.

A sample fault policy file is shown in [Example 21-1](#):

Example 21-1 Sample Fault Policy File

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicies>
  <faultPolicy version="2.0.1" id="CRM_ServiceFaults">
    <Conditions>
      <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
        <condition>
          <test>contains($fault.mediatorErrorCode, "TYPE_FATAL_MESH")</test>
          <action ref="ora-retry"/>
        </condition>
      </faultName>
    </Conditions>
    <Actions>
      <Action id="ora-retry">
        <retry>
          <retryCount>3</retryCount>
          <retryInterval>2</retryInterval>
          <exponentialBackoff/>
          <retryFailureAction ref="ora-java"/>
          <retrySuccessAction ref="ora-terminate"/>
        </retry>
      </Action>
    </Actions>
  </faultPolicy>
</faultPolicies>
```

The two components of the fault policy (conditions and actions) are described in the following sections.

21.1.1.1 Conditions

Conditions identify error or fault conditions along with a 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 a retry. Similarly, for another error occurring because of the failure of Schematron validation, you can perform the action of human intervention. This fault can be recovered manually by editing the payload and then resubmitting it through Oracle Enterprise Manager Fusion Middleware Control.

Conditions are defined in the `fault-policies.xml` file, as shown in [Example 21-2](#):

Example 21-2 Conditions

```
<Conditions>
  <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
```

```

        name="medns:mediatorFault">
            <condition>
                <test>contains($fault.mediatorErrorCode, "TYPE_DATA_
TRANSFORMATION")</test>
                <action ref="ora-java"/>
            </condition>
        </faultName>
        <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
            <condition>
                <test>contains($fault.mediatorErrorCode, "TYPE_FATAL_MESH")</test>
                <action ref="ora-retry"/>
            </condition>
        </faultName>
        <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
            <condition>
                <test>contains($fault.mediatorErrorCode, "TYPE_DATA_ASSIGN")</test>
                <action ref="ora-retry-crm-endpoint"/>
            </condition>
        </faultName>
    </Conditions>

```

Identifying Fault Types Using Conditions

You can categorize the faults that can be captured using conditions into the following types:

- Oracle Mediator-specific faults

For all Oracle Mediator-specific faults, the Oracle Mediator service engine throws only one fault, namely

{http://schemas.oracle.com/mediator/faults}mediatorFault.

Every Oracle Mediator fault is wrapped into this fault. The errors or faults generated by an Oracle Mediator can be captured by using the format shown in [Example 21-3](#):

Example 21-3 Oracle Mediator-Specific Faults

```

<faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
    <!-- mediatorFault is a bucket for all the mediator faults -->
    <condition>
        <test>
            contains($fault.mediatorErrorCode, "TYPE_FATAL_MESH")
        </test>
    <!-- Captures TYPE_FATAL_MESH errors -->
        <action ref="ora-retry"/>
    </condition>
</faultName>

```

- Business faults and SOAP faults

These errors or faults can be captured by defining an XPath condition, which is based on the fault payload. [Example 21-4](#) provides details.

Example 21-4 Business Faults and SOAP Faults

```

<faultName xmlns:ns1="http://xmlns.oracle.com/Customer"
name="ns1:InvalidCustomer"> <!-- QName of Business/SOAP fault -->
    <condition>

```

```

        <test>
contains($fault.<PART_NAME>/custid, 1011)
        </test>
<!-- xpath condition based on fault payload -->
        <action ref="ora-retry"/>
    </condition>
</faultName>

```

When a reference service returns a business fault, the fault can be handled in the Oracle Mediator service component. The returned fault can be forwarded to another component, redirected to an adapter service such as a file adapter, or an event can be raised. However, if both a fault policy and fault handler are defined for a business fault, then the fault policy takes precedence over the fault handler. In such a case, the fault handlers in the Oracle Mediator service component are ignored, if the fault policy is successfully executed.

- **Adapter-specific fault**

The errors or faults generated by an adapter can be captured by using the format shown in [Example 21–5](#):

Example 21–5 Capturing Errors or Faults Generated by an Adapter

```

<faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
    <condition>
        <test>$fault.faultCode = "1"</test> <!-- unique constraint violation in DB
adapter-->
        <action ref="ora-retry"/>
    </condition>
</faultName>

```

21.1.1.2 Actions

Actions specify the tasks to perform 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:

- **Retry:** Retry actions such as enqueueing a message to a JMS queue that is stopped, inserting a record with a unique key constraint error, and so on, enable you to retry a task that caused the error. A new thread is started with every retry action. Therefore, with every retry action, a new transaction starts. [Table 21–1](#) describes the options available with the retry action.

Table 21–1 Retry Action Options

Option	Description
Retry Count	Retry <i>N</i> times.
Retry Interval	Delay in seconds for a retry.
Exponential Backoff	Retry interval increase with an exponential backoff.
Retry Failure Action	Chain to this action if a retry <i>N</i> times fails.
Retry Success Action	Chain to this action if a retry succeeds.

Note: Exponential backoff indicates that the next retry attempt is scheduled at $2 \times$ the *delay*, where *delay* is the current retry interval. For example, if the current retry interval is 2 seconds, the next retry attempt is scheduled at 4, the next at 8, and the next at 16 seconds until the `retryCount` value is reached.

Example 21-6 shows how to specify the retry action:

Example 21-6 Retry Action

```
<Action id="ora-retry">
  <retry>
    <retryCount>3</retryCount>
    <retryInterval>2</retryInterval>
    <exponentialBackoff/>
    <retryFailureAction ref="ora-java"/>
    <retrySuccessAction ref="ora-java"/>
  </retry>
</Action>
```

If you set the retry Interval in the fault policy to a duration of less than 30 seconds, then the retry may not happen within the specified intervals. This is because the default value of the `org.quartz.scheduler.idleWaitTime` property is 30 seconds, and the scheduler waits for 30 seconds before retrying for available triggers, when the scheduler is otherwise idle. If the retry interval is set to a value of less than 30 seconds, then latency is expected.

If you want the system to use a retry interval that is less than 30 seconds, add the following property under the section `<property name="quartzProperties">` in the `fabric-config-core.xml` file:

```
org.quartz.scheduler.idleWaitTime=<value>
```

- Human intervention: You can specify this action in the following way:

```
<Action id="ora-human-intervention"><humanIntervention/></Action>
```

- Abort: This action enables you to abort the flow. You can specify this action in the following way:

```
<Action id="ora-terminate"><abort/></Action>
```

- Java code: This action enables you to call a customized Java class that implements the `oracle.integration.platform.faultpolicy.IFaultRecoveryJavaClass` interface. You can specify this action as shown in Example 21-7:

Note: The implemented Java class must implement a method that returns a string. The policy can be chained to a new action based on the returned string.

Example 21-7 Customized Java Class Calling

```
<Action id="ora-java">
  <javaAction className="mypackage.myClass" defaultAction="ora-terminate">
    <returnValue value="ABORT" ref="ora-terminate"/>
    <returnValue value="RETRY" ref="ora-retry"/>
  </javaAction>
</Action>
```

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

For more information, see [Example 21–8](#) and [Example 21–9](#).

Example 21–8 *oracle.integration.platform.faultpolicy.IFaultRecoveryJavaClass Interface*

```
oracle.integration.platform.faultpolicy.IFaultRecoveryJavaClass {

    public void handleRetrySuccess(IFaultRecoveryContext ctx);
    public String handleFault(IFaultRecoveryContext ctx);

}
```

Example 21–9 *oracle.integration.platform.faultpolicy.IFaultRecoveryContext Interface*

```
public interface IFaultRecoveryContext {

    /**
     * Gets implementation type of the fault.
     * @return
     */
    public String getType();

    /**
     * @return Get property set of the fault policy action being executed.
     */
    public Map getProperties();

    /**
     * @return Get fault policy id of the fault policy being executed.
     */
    public String getPolicyId();

    /**
     * @return Name of the faulted reference.
     */
    public String getReferenceName();

    /**
     * @return Port type of the faulted reference link.
     */
    public QName getPortType();

}
```

Oracle Mediator Service Engine Implementation

[Example 21–10](#) shows the Oracle Mediator service engine implementation of the `IFaultRecoveryContext` interface.

Example 21–10 *IFaultRecoveryContext Interface Implementation*

```
package oracle.tip.mediator.common.error.recovery;
public class MediatorRecoveryContext implements IFaultRecoveryContext{
    ...
}
```

You can use the methods shown in [Example 21–11](#) to retrieve additional Oracle Mediator-specific data available with the `MediatorRecoveryContext` class:

Example 21–11 Methods for Retrieving Data

```
public Fault getFault()
public CalloutMediatorMessage getMediatorMessage()
```

[Example 21–12](#) shows how to retrieve data using the `CalloutMediatorMessage` interface:

Example 21–12 Data Retrieval Using the `CalloutMediatorMessage` Interface

```
/**
 * Accessing Mediator Message properties by providing the property name
 * @param propertyName
 * @return
 * @throws MediatorException
 */
public Object getProperty(String propertyName);

/**
 * Accessing Mediator Message properties
 * @return
 * @throws MediatorException
 */
public Map getProperties();

/**
 * Accessing instance id of the mediator message
 * This will be the mediator instance id created for that particular message
 * object
 * @return
 * @throws MediatorException
 */
public String getId() throws MediatorException;

/**
 * Accessing payload of the mediator message
 * object
 * @return
 * @throws MediatorException
 */
public Map getPayload();

/**
 * Accessing header of the mediator message
 * object
 * @return
 * @throws MediatorException
 */
public List<Element> getHeaders();

/**
 * Accessing componentDN for mediator component
 * @return
 * @throws MediatorException
 */
public String getComponentDN()

/**
 * Setting payload to the mediator message
 * @return
 * @throws MediatorCalloutException
 */
```

```
    public void addPayload(String partName, Object payload) throws
MediatorCalloutException;

    /**
     * Adding property to the mediator message
     * @return
     * @throws MediatorCalloutException
     */
    public void addProperty(String name, Object value) throws
MediatorCalloutException;

    /**
     * Adding header to the mediator message
     * @return
     * @throws MediatorCalloutException
     */
    public void addHeader(Object header) throws MediatorCalloutException;
```

21.1.2 Fault Bindings

Fault bindings associate fault policies with composites or components, and are defined in the `fault-bindings.xml` file. Create the `fault-bindings.xml` file based on the XML schema defined in [Section 21.4.2, "Schema Definition File for fault-bindings.xml."](#)

Fault policies can be created at the following levels:

- **Composite:** You can define one fault policy for all Oracle Mediator components in a composite. You can specify this level in the following way:

```
<composite faultPolicy="ConnectionFaults"/>
```

- **Component:** You can define a fault policy exclusively for an Oracle Mediator service component. A component-level fault policy overrides the composite-level fault policy. You can specify this level as shown in [Example 21–13](#).

Example 21–13 Definition of a Fault Policy for an Oracle Mediator

```
<component faultPolicy="ConnectionFaults">
    <name>Component1</name>
    <name>Component2</name>
</component>
```

- **Reference:** You can define a fault policy for the references of an Oracle Mediator component. You can specify this level as shown in [Example 21–14](#).

Example 21–14 Definition of a Fault Policy for a Reference

```
<reference faultPolicy="policy1">
    <name>DBAdapter3</name>
</reference>
```

Note: Human intervention is the default action for errors that do not have a fault policy defined.

A sample fault binding file is shown in [Example 21–15](#).

Example 21–15 Sample Fault Binding File

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicyBindings version="2.0.1"
  xmlns="http://schemas.oracle.com/bpel/faultpolicy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <composite faultPolicy="ConnectionFaults"/>
</faultPolicyBindings>
```

21.1.3 Error Groups in Oracle Mediator

You can specify an action for an error type or error group while defining the conditions in a fault policy. In the previous examples, `medns:mediatorFault` indicates that the error is an Oracle Mediator error, whereas `medns:TYPE_FATAL_MESH` refers to an error group. An error group consists of one or more child error types. `TYPE_ALL` is an error group that contains all Oracle Mediator errors.

The following list describes various error groups contained in the `TYPE_ALL` error group:

- `TYPE_DATA`: Contains errors related to data handling.
 - `TYPE_DATA_ASSIGN`: Contains errors related to data assignment.
 - `TYPE_DATA_FILTERING`: Contains errors related to data filtering.
 - `TYPE_DATA_TRANSFORMATION`: Contains errors that occur during a transformation.
 - `TYPE_DATA_VALIDATION`: Contains errors that occur during payload validation.
- `TYPE_METADATA`: Contains errors related to Oracle Mediator metadata.
 - `TYPE_METADATA_FILTERING`: Contains errors that occur while processing the filtering conditions.
 - `TYPE_METADATA_TRANSFORMATION`: Contains errors that occur while getting the metadata for a transformation.
 - `TYPE_METADATA_VALIDATION`: Contains errors that occur during validation of metadata for Oracle Mediator (.mpplan 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 a `Datasource not found` error.
 - `TYPE_FATAL_CACHE`: Contains Oracle Mediator cache-related fatal errors.
 - `TYPE_FATAL_ERRORHANDLING`: Contains fatal errors that occur during error handling such as `Resubmission queues not available`.
 - `TYPE_FATAL_MESH`: Contains fatal errors from the Service Infrastructure such as `Invoke service not available`.
 - `TYPE_FATAL_MESSAGING`: Contains fatal messaging errors arising from the Service Infrastructure.
 - `TYPE_FATAL_TRANSACTION`: Contains fatal errors related to transactions such as `Commit can't be called on a transaction which is marked for rollback`.

- `TYPE_FATAL_TRANSFORMATION`: Contains fatal transformation errors such as an error occurring because of the XPath functions used in a transformation.
- `TYPE_TRANSIENT`: Contains transient errors that can be recovered on a retry.
 - `TYPE_TRANSIENT_MESH`: Contains errors related to the Service Infrastructure.
 - `TYPE_TRANSIENT_MESSAGING`: Contains errors related to JMS such as enqueueing and dequeuing.
- `TYPE_INTERNAL`: Contains internal errors.

21.2 Using Error Handling with Oracle Mediator

You can enable error handling for an Oracle Mediator by using the `fault-policies.xml` and `fault-bindings.xml` files.

21.2.1 How to Use Error Handling for an Oracle Mediator Service Component

To use error handling for an Oracle Mediator service component:

1. Create a `fault-policies.xml` file based on the schema defined in [Section 21.4.1, "Schema Definition File for fault-policies.xml."](#)
2. Create a `fault-bindings.xml` file based on the schema defined in [Section 21.4.2, "Schema Definition File for fault-bindings.xml."](#)
3. Copy the `fault-policies.xml` and the `fault-bindings.xml` file to your SOA composite application project directory.
4. Deploy the SOA composite application project.

21.2.2 What Happens at Runtime

All the fault policies for a composite are loaded when the first error occurs. At runtime, Oracle Mediator checks whether there is any policy defined for the current error. If a fault policy is defined, then Oracle Mediator performs the action according to the configuration in the fault policies file. If there is no fault policy defined, then the default action of human intervention is performed.

21.3 Fault Recovery Using Oracle Enterprise Manager Fusion Middleware Control

Apart from policy-based recovery using the fault policy file, you can also perform fault recovery actions on Oracle Mediator faults identified as recoverable in Oracle Enterprise Manager Fusion Middleware Control. This can be performed in the following ways:

- Manual recovery by modifying the payload using Oracle Enterprise Manager Fusion Middleware Control
- Bulk recovery without modifying the payload using Oracle Enterprise Manager Fusion Middleware Control
- Aborting a faulted instance using Oracle Enterprise Manager Fusion Middleware Control, if you do not want to do any more processing on the instance.

For more information about fault recovery using Oracle Enterprise Manager Fusion Middleware Control, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

21.4 Error Handling XML Schema Definition Files

This section describes the schema files for the `fault-policies.xml` and `fault-bindings.xml` files.

21.4.1 Schema Definition File for `fault-policies.xml`

The `fault-policies.xml` file should be based on the XSD file shown in [Example 21-16](#).

Example 21-16 XSD File for `fault-policies.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schemas.oracle.com/bpel/faultpolicy"
  xmlns:tns="http://schemas.oracle.com/bpel/faultpolicy"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <!-- Conditions contain a list of fault names -->
  <xs:element name="Conditions">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="faultName" type="tns:faultNameType"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- action Ref must exist in the same file -->
  <xs:complexType name="actionRefType">
    <xs:attribute name="ref" type="xs:string" use="required"/>
  </xs:complexType>
  <!-- one condition has a test and action, if test is missing, this is the
  catch all condition -->
  <xs:complexType name="conditionType">
    <xs:all>
      <xs:element name="test" type="tns:idType" minOccurs="0"/>
      <xs:element name="action" type="tns:actionRefType"/>
    </xs:all>
  </xs:complexType>
  <!-- One fault name match contains several conditions -->
  <xs:complexType name="faultNameType">
    <xs:sequence>
      <xs:element name="condition" type="tns:conditionType"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:QName"/>
  </xs:complexType>
  <xs:complexType name="ActionType">
    <xs:choice>
      <xs:element name="retry" type="tns:RetryType"/>
      <xs:element ref="tns:rethrowFault"/>
      <xs:element ref="tns:humanIntervention"/>
      <xs:element ref="tns:abort"/>
      <xs:element ref="tns:replayScope"/>
      <xs:element name="javaAction" type="tns:JavaActionType">
        <xs:key name="UniqueReturnValue">
          <xs:selector xpath="tns:returnValue"/>

```

```

        <xs:field xpath="@value"/>
    </xs:key>
</xs:element>
</xs:choice>
<xs:attribute name="id" type="tns:idType" use="required"/>
</xs:complexType>
<xs:element name="Actions">
    <xs:annotation>
        <xs:documentation>Fault Recovery Actions</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Action" type="tns:ActionType"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:complexType name="JavaActionType">
    <xs:annotation>
        <xs:documentation>This action invokes java code
provided</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="returnValue" type="tns:ReturnValueType"
minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="className" type="tns:idType" use="required"/>
    <xs:attribute name="defaultAction" type="tns:idType" use="required"/>
    <xs:attribute name="propertySet" type="tns:idType"/>
</xs:complexType>
<xs:complexType name="RetryType">
    <xs:annotation>
        <xs:documentation>This action attempts retry of activity
execution</xs:documentation>
    </xs:annotation>
    <xs:all>
        <xs:element ref="tns:retryCount"/>
        <xs:element ref="tns:retryInterval"/>
        <xs:element ref="tns:exponentialBackoff" minOccurs="0"/>
        <xs:element name="retryFailureAction"
type="tns:retryFailureActionType" minOccurs="0"/>
        <xs:element name="retrySuccessAction"
type="tns:retrySuccessActionType" minOccurs="0"/>
    </xs:all>
</xs:complexType>
<xs:simpleType name="idType">
    <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="ReturnValueType">
    <xs:annotation>
        <xs:documentation>Return value from java code can chain another action
using
        return values</xs:documentation>
    </xs:annotation>
    <xs:attribute name="value" type="tns:idType" use="required"/>
    <xs:attribute name="ref" type="xs:string" use="required"/>
</xs:complexType>
<xs:element name="exponentialBackoff">

```



```

        <xs:annotation>
            <xs:documentation>Setting this will cause retry attempts to use
                exponentialBackoff algorithm</xs:documentation>
        </xs:annotation>
    </xs:complexType/>
</xs:element>
<xs:element name="humanIntervention">
    <xs:annotation>
        <xs:documentation>This action causes the activity to
freeze</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
</xs:element>
<xs:element name="replayScope">
    <xs:annotation>
        <xs:documentation>This action replays the immediate enclosing
scope</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
</xs:element>
<xs:element name="rethrowFault">
    <xs:annotation>
        <xs:documentation>This action will rethrow the
fault</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
</xs:element>
<xs:element name="retryCount" type="xs:positiveInteger">
    <xs:annotation>
        <xs:documentation>This value is used to identify number of
retries</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="retryFailureActionType">
    <xs:annotation>
        <xs:documentation>This is the action to be chained if retry attempts
fail</xs:documentation>
    </xs:annotation>
    <xs:attribute name="ref" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="retrySuccessActionType">
    <xs:annotation>
        <xs:documentation>This is the action to be chained if retry attempts
is successful</xs:documentation>
    </xs:annotation>
    <xs:attribute name="ref" type="xs:string" use="required"/>
</xs:complexType>
<xs:element name="retryInterval" type="xs:unsignedLong">
    <xs:annotation>
        <xs:documentation>This is the delay in milliseconds of retry
attempts</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="abort">
    <xs:annotation>
        <xs:documentation>This action terminates the
process</xs:documentation>
    </xs:annotation>
    <xs:complexType/>
</xs:element>

```

```

    <xs:element name="Properties">
      <xs:annotation>
        <xs:documentation>Properties that can be passes to a custom java
class</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="propertySet" type="tns:PropertySetType"
maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:complexType name="PropertySetType">
      <xs:sequence>
        <xs:element name="property" type="tns:PropertyValueType"
maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="tns:idType" use="required" />
    </xs:complexType>
    <xs:complexType name="PropertyValueType">
      <xs:simpleContent>
        <xs:extension base="tns:idType">
          <xs:attribute name="name" type="tns:idType" use="required" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
    <xs:element name="faultPolicy">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:Conditions" />
          <xs:element ref="tns:Actions" />
          <xs: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:element>
  </xs:sequence>
</xs:complexType>

```

```

        </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>

```

21.4.2 Schema Definition File for fault-bindings.xml

The `fault-bindings.xml` file should be based on the XSD file shown in [Example 21–17](#).

Example 21–17 XSD File for fault-bindings.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schemas.oracle.com/bpel/faultpolicy"
xmlns:tns="http://schemas.oracle.com/bpel/faultpolicy"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="faultPolicyBindings">
        <xs:annotation>
            <xs:documentation>Bindings to a specific fault policy
        </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="composite" type="tns:compositeType"
minOccurs="0" maxOccurs="1"/>
                <xs:element name="component" type="tns:componentType"
minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="reference" type="tns:referenceType"
minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="version" type="xs:string" default="2.0.1"/>
        </xs:complexType>
        <xs:key name="UniquePartnerLinkName">
            <xs:selector xpath="tns:reference/tns:name"/>
            <xs:field xpath="."/>
        </xs:key>
        <xs:key name="UniquePortType">
            <xs:selector xpath="tns:reference/tns:portType"/>
            <xs:field xpath="."/>
        </xs:key>
        <xs:key name="UniquePolicyName">

```

```
        <xs:selector xpath="tns:reference"/>
        <xs:field xpath="@faultPolicy"/>
    </xs:key>
</xs:element>
<xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="propertyType">
    <xs:simpleContent>
        <xs:extension base="tns:nameType">
            <xs:attribute name="name" type="xs:string" use="required"
fixed="faultPolicy"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="referenceType">
    <xs:annotation>
        <xs:documentation>Bindings for a partner link. Overrides composite
level binding.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:annotation>
            <xs:documentation>Specification at partner link name overrides
specification for a port type</xs:documentation>
        </xs:annotation>
        <xs:element name="name" type="tns:nameType" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="portType" type="xs:QName" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="faultPolicy" type="tns:nameType" use="required"/>
</xs:complexType>

<xs:complexType name="componentType">
    <xs:annotation>
        <xs:documentation>Binding for a component </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="name" type="tns:nameType" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="faultPolicy" type="tns:nameType" use="required"/>
</xs:complexType>
<xs:complexType name="compositeType">
    <xs:annotation>
        <xs:documentation>Binding for the entire composite</xs:documentation>
    </xs:annotation>
    <xs:attribute name="faultPolicy" type="tns:nameType" use="required"/>
</xs:complexType>
</xs:schema>
```

Resequencing in Oracle Mediator

This chapter describes support for message resequencing in Oracle Mediator. It contains the following sections:

- [Section 22.1, "Introduction to the Resequencer"](#)
- [Section 22.2, "Resequencing Order"](#)
- [Section 22.3, "Configuring the Resequencer"](#)
- [Section 22.4, "Limitations in the Resequencer"](#)

22.1 Introduction to the Resequencer

The resequencer in Oracle Mediator rearranges a stream of related but out-of-sequence messages into a sequential order. When incoming messages arrive, they may be in a random order. The resequencer orders the messages based on sequential or chronological information, and then sends the message to the target services in an orderly manner. The sequencing is performed based on the sequencing strategy selected.

22.1.1 Groups and Sequence IDs

The resequencer works with two central concepts: groups and sequence IDs. The sequence ID is an identifying part of the message, and messages are rearranged based on this identifier. The messages arriving for resequencing are split into groups and the messages within a group are sequenced according to the sequence ID. Sequencing within a group is independent of the sequencing of messages in any other group. Groups in themselves are not dependent on each other and can be processed independently of each other.

As an example, messages attached to certain groups arrive to an Oracle Mediator service component in the following order:

msg9(a), msg8(b), msg7(a), msg6(c), msg5(a), msg4(b), msg3(c), msg2(b), msg1(a)

[Table 22–1](#) shows how the Oracle Mediator sorts the messages into groups. The order of the messages in each group depends on the type of resequencer used.

Table 22–1 Messages Sorted into Groups

Group c	Group b	Group a
msg6(c), msg3(c)	msg8(b), msg4(b), msg2(b)	msg9(a), msg7(a), msg5(a), msg1(a)

All the groups are processed independently of each other and any error occurring in one group does not affect the processing of other groups.

22.1.2 Identification of Groups and Sequence IDs

Groups and sequence IDs are identified through XPath expressions in the payload. You specify XPath expressions that point to the elements in the message payload on which grouping is done and on which sequencing is done.

In the message payload shown in [Figure 22–1](#), `CustomerId` is the field on which to base instance sequencing and `CustomerName` is the field on which to base grouping.

Figure 22–1 Message Payload

```
<?xml version="1.0" encoding="UTF-8"?>
<CU:CustomerData
  xmlns:CU="http://xmlns.oracle.com/Esb/CustomerData"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <CustomerId>1</CustomerId>
  <CustomerName>Group</CustomerName>
  <Type>Gold</Type>
  <Description>Accounting outsourcing Partner</Description>
  <Address>3228 Massilon Blvd</Address>
  <City>Juniper</City>
  <State>Massachusetts</State>
  <Zip>01854</Zip>
  <Country>US</Country>
  <Phone>877-555-9876</Phone>
  <Status>Active</Status>
  <CreditRating>5</CreditRating>
  <Discount>0</Discount>
  <Terms>30n4</Terms>
  <EnrollDate>01/1/01</EnrollDate>
  <LastOrderDate>05/05/05</LastOrderDate>
  <Currency>USD</Currency>
  <ContactName>Jan Forester</ContactName>
  <ContactTitle>VP Finance</ContactTitle>
  <ContactPhone>877-555-9000</ContactPhone>
  <AccountRep>Geoff Seattle</AccountRep>
  <CampaignRating>2</CampaignRating>
  <ReferredBy>Houston America Taxco</ReferredBy>
</CU:CustomerData>
```

Note: Resequencing is supported only for Oracle Mediator components that have a request operation and a request-callback operation in the WSDL file. In other words, resequencing is not allowed by the user interface if the WSDL operation has a synchronous reply element. For more information about these operations, see [Chapter 23, "Understanding Message Exchange Patterns of an Oracle Mediator."](#)

22.2 Resequencing Order

Oracle Mediator can resequence the incoming messages in a user-specified order. This implementation enables you to specify three types of resequencing orders:

- [Standard Resequencer](#)
- [FIFO Resequencer](#)
- [Best Effort Resequencer](#)

22.2.1 Standard Resequencer

The standard resequencer supports a standard resequencer pattern. The following sections describe the standard resequencer and how it processes messages.

22.2.1.1 Overview of the Standard Resequencer

The standard resequencer is useful for applications that use identifiers from a simple numeric identifier sequence in their messages. The standard resequencer receives a stream of messages that might not arrive in order; it then stores the out-of-sequence messages until a complete sequence based on the sequence IDs is received. The in-sequence messages are then processed asynchronously based on their sequence ID.

It is important to note that the messages to outbound services of the standard resequencer Oracle Mediator service component are guaranteed to arrive in sequence. The standard resequencer does not modify the message contents; it just orders them.

22.2.1.2 Information Required for Standard Resequencing

When using the standard resequencer in Oracle Mediator, you must always specify a group XPath expression and a sequence ID XPath expression. These specify where the Oracle Mediator resequencer can find the group and the sequence ID in the messages. You must also supply the sequence numbering in terms of the start sequence ID and the sequence ID incremental delta. This numbering is used to form each group. In addition to the group, sequence ID, and increment properties, you can also specify a time period, in seconds, to wait for the expected messages.

22.2.1.3 Example of the Standard Resequencer

[Table 22–2](#) shows how groups are formed differently for two different values of the incremental delta.

Table 22–2 Groups Formed Differently for Two Different Values

Start SequenceID	Incremental Delta	Group1	Group2	...	Groupn
1	1	1,2,3,4,5,...	1,2,3,4,5,...	...	1,2,3,4,5,...n
1	5	1,5,10,15,...	1,5,10,15,...	...	1,5,10,15,...

Notes:

- If the sequence numbering is different for various groups (for example, if the groups do not have the same incremental delta or start sequence ID) and the messages do not arrive in order, then you can use the best effort resequencer to rearrange the messages.
- The Oracle Mediator standard resequencer holds back messages in the Oracle Mediator resequencer database until it can produce the right sequence for different groups. This means that if for a given group, the message with a particular sequence ID does not arrive within the timeout period¹, the consecutive messages for that group are held back forever. In such a case, you must manually unlock the group through Oracle Enterprise Manager Fusion Middleware Control and go to the next available message, skipping the pending message.

¹ The timeout period is the time period in seconds to wait for an expected message.

22.2.2 FIFO Resequencer

The FIFO resequencer supports a standard first in, first out (FIFO) pattern. The following sections describe the FIFO resequencer and how it processes messages.

22.2.2.1 Overview of the FIFO Resequencer

The FIFO resequencer is useful for applications that need sequencing based on the time the messages arrive to the Oracle Mediator. The FIFO resequencer receives a stream of messages that are in order and processes them in sequence for each group based on the arrival time of the messages.

It is important to note that the messages to outbound services of the Oracle Mediator acting as a FIFO resequencer are guaranteed to arrive in order based on arrival time. Therefore, the messages are delivered in the order they were stored in the resequencer data store.

22.2.2.2 Information Required for FIFO Resequencing

When using the FIFO resequencer, you must always specify a group XPath expression. However, you do not need to specify a sequence ID because the messages are processed according to the time of arrival to the Oracle Mediator service component that is configured for FIFO resequencing. The group XPath expression specifies where the FIFO resequencer should find the group information in the message to group the messages. No further configuration is needed for a FIFO pattern.

22.2.2.3 Example of the FIFO Resequencer

[Table 22–3](#) illustrates the behavior of the FIFO resequencer where `msgX(Y, Z)` indicates that the message arrives as message number X to the Oracle Mediator service component and the message contains `sequenceID` Y and group Z.

Table 22–3 *FIFO Resequencer Behavior*

Incoming Messages	Sequenced Messages
msg03(2,c)	msg12(4,c),msg10(3,c),msg06(1,c),msg03(2,c)
msg06(1,c)	msg05(9,a), msg02(7,a), msg10(3,a), msg07(5,a)
msg07(5,a)	
msg10(3,a)	
msg10(3,c)	
msg02(7,a)	
msg05(9,a)	
msg12(4,c)	

As shown in [Table 22–3](#), the messages are sequenced based on their time of arrival and the `sequenceID` is not used for sequencing.

Note: When using the FIFO resequencer, use a single-threaded inbound adapter to avoid unpredictable results. For example, when you use the file/FTP adapter, the database adapter, or the AQ adapter in front of an Oracle Mediator service component that is configured as a FIFO resequencer, configure the adapter for single-threaded processing. Otherwise, unpredictable results occur because the arrival time of each message is calculated when the message arrives to the Oracle Mediator service component instead when it arrives to the adapter service.

22.2.3 Best Effort Resequencer

The Oracle Mediator resequencer supports a best effort pattern. The following sections describe the best effort resequencer and how it processes messages.

22.2.3.1 Overview of the Best Effort Resequencer

The best effort pattern is useful for applications that produce a large number of messages in a short period of time and cannot provide information to the resequencer about the identifier to use for sequencing. Typically, the identifier used for sequencing in such scenarios is of a `dateTime` type or `numeric` type. Using the `dateTime` field as the sequence ID XPath enables you to control the sequencing. The messages are expected to be sent in sequence by the applications, thus the date and time the messages are sent can be used for sequencing. The Oracle Mediator makes the best effort to ensure that the messages are delivered in sequence.

The best effort resequencer can reorder messages based on no knowledge about the increment of the sequence ID. This means that unlike the standard resequencer, you do not need to define the increment of the sequence ID for the best effort resequencer in advance. When the messages are processed, they are processed in sequence based on the specified sequence ID and the messages that have arrived, whether a true sequence is received. The sequence IDs are either `numeric` or `dateTime`. Therefore, sequencing occurs on the numeric order or the `dateTime` order of the sequence IDs.

22.2.3.1.1 Best Effort Resequencer Message Selection Strategies

The best effort resequencer processes messages asynchronously based on one of two message selection strategies: Maximum rows selected or time window. The messages selected and processed at any one time are based either on the maximum number of rows you specify or on a window of time in which they arrive.

Maximum Rows Selected

When the best effort resequencer is configured to use a maximum number of rows, it performs the following steps whenever new messages are available in the resequencer database:

1. The resequencer orders the messages according to the specified sequence ID (typically a date and time stamp).
2. The resequencer locks and selects the number of messages equal to the value of the `maxRowsRetrieved` parameter from the ordered messages above.
3. The resequencer processes the selected messages one after another in its own transaction in sequence.

Time Window

When the best effort resequencer is configured to use a time window instead of a maximum number rows, the messages to select and process at one time are based on a period of time you specify plus an optional buffer time. Each message belongs to a specific time window, and messages that are part of one time window are processed separately from messages belonging to a different time window.

In addition to the time window, you can specify a buffer time, which is an overlap between two sequential time windows that allows messages that arrive a little late to be associated with the first time window. By default, the buffer time is 10% of the time window you specify.

When the best effort resequencer is configured to use a time window, groups are processed in an iterative manner and messages are processed in the following steps:

1. The first message arrives and the time window begins.
2. The buffer is added to the time window, and processing begins after the buffer time.
3. The resequencer retrieves the messages that arrived within the time window, and identifies the maximum sequence ID (typically a date and time stamp) from all the messages.
4. The resequencer retrieves any messages that arrive within the buffer time and that have a sequence ID that is less than the maximum sequence ID identified above.
5. The resequencer sorts all messages retrieved in the above steps in ascending order of the sequence IDs and processes the messages.

22.2.3.1.2 Best Effort Resequencer Message Delivery

It is important to note that the messages to outbound services of the Oracle Mediator service component configured for best effort resequencing are not guaranteed to arrive in order of a sequence ID. At any given time, a snapshot of the available messages is taken and sequencing is performed only on those messages. Therefore, unlike a standard resequencer, it is not guaranteed that a message with a lesser sequence ID value is sent before a message that has a greater sequence ID value but that arrived earlier. Messages with a lesser sequence ID value that arrive later might be processed in the following cycle when a snapshot of available messages is taken again and the messages are reordered.

22.2.3.2 Information Required for Best Effort Resequencing

When using the best effort resequencer, you must specify a group XPath expression, a sequence ID XPath expression, and the date type of the sequence ID (`numeric` or `dateTime`). These specify where the resequencer should find the group and the sequence ID in the messages and how to handle the sequence ID. In addition, you must specify either a maximum number of rows to select for each resequencing batch or a time window during which the messages included in one batch arrive.

Unlike the standard resequencer, the best effort resequencer has no knowledge about how the sequence is built. No further information is used by the best effort resequencer to perform its responsibilities.

22.2.3.3 Example of Best Effort Resequencing Based on Maximum Rows

[Table 22–4](#) illustrates the behavior of the best effort resequencer when it is configured to use the maximum number of rows to determine which messages to process. In this example, `msgX (Y, Z)` indicates that the message arrives as message number X to the

Oracle Mediator service component and the message contains sequenceID Y and group Z.

Table 22–4 Best Effort Resequencer Behavior Based on Maximum Rows

Group C	Sequenced Messages
msg03(1,c)	msg12(4,c),msg10(3,c),msg06(2,c),msg03(1,c)
msg06(2,c)	
msg10(3,c)	
msg12(4,c)	

Note: For the best effort resequencer to work correctly, the messages must arrive in sequence or nearly in sequence. Otherwise, they are not resequenced correctly. If the messages do not arrive close together, set the value of the `maxRowsRetrieved` parameter to 1 so the next message in the sequence has enough time to arrive and be picked up by the next processing loop (and therefore be delivered in sequence).

22.2.3.4 Example of Best Effort Resequencing Based on a Time Window

Table 22–5 illustrates the behavior of the best effort resequencer when it is configured to process messages based on the time period in which they arrive. In this example, the time window is 10 minutes, the buffer is 10% (one minute), and `msgX (Y)` indicates that the message arrives as message number X to the Oracle Mediator service component and the message contains the sequence ID Y. The first message arrives at 2:00:00, which starts the time window. The time window lasts until 2:10:00, but with the addition of the buffer time, messages that arrived until 2:11:00 are processed.

Table 22–5 Best Effort Resequencer Behavior Based on a Time Window

Group C	
Message/Time	Sequenced Messages
msg01(04)/2:00:00	msg03(01), msg06(02), msg04(03), msg01(04), msg02(05),
msg02(05)/2:00:20	msg09(06), msg05(07), msg08(08), msg12(09), msg11(10),
msg03(01)/2:00:30	msg10(12), msg07(13)
msg04(03)/2:00:50	
msg05(07)/2:04:20	
msg06(02)/2:04:45	
msg07(13)/2:05:10	
msg08(08)/2:05:40	
msg09(06)/2:08:40	
msg10(12)/2:09:20	
msg11(10)/2:10:30	
msg12(09)/2:10:40	
msg13(14)/2:10:50	
msg14(11)/2:13:00	

Note: In the above example, the resequencer identified the maximum sequence ID for the time window as 13 (from message 7). Message 13 arrived within the buffer time, but has a sequence ID of 14. It is not processed with the original group, but instead begins a new time window at its arrival time of 2:10:50. Message 14 arrived too late and is included in the second time window.

22.3 Configuring the Resequencer

You can configure the resequencer using Oracle JDeveloper. This section describes how to configure the resequencer in Oracle JDeveloper.

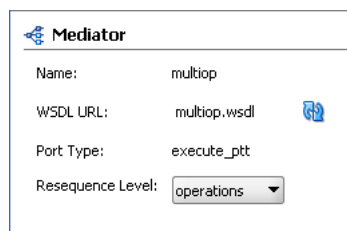
22.3.1 How to Specify the Resequencing Level

You can define resequencing at either the service component level or the operation level. For Oracle Mediator service components with only one operation, configuring resequencing at the operation or service component level results in the same behavior. For Oracle Mediator service components having multiple operations, specifying the resequencing at the service component level applies the same resequencing rules to all the operations, and messages arriving at any operation are resequenced. By default, the resequencing level is **operations**.

To specify the resequencing level:

1. On the Mediator Editor, select the resequencing level from the **Resequence Level** dropdown list, as shown in [Figure 22–2](#).

Figure 22–2 Mediator Editor with Resequence Level Field



If you choose **component**, the **Resequence** field under each operation no longer appears and the **Resequence Mode** field appears under the **Resequence Level** field so you can set the resequencing mode for the service component. By default, the resequencing mode is set to **off**.

When you select a resequencing mode, the **Resequence Options** box appears under the service component or operation, as shown in [Figure 22–3](#). If the **Resequence Mode** field for an operation is set to **off**, the **Resequence Options** box disappears.

Figure 22–3 Mediator Editor with Resequence Options Section

 Mediator

Name:multiop

WSDL URL:multiop.wsdl

Port Type:execute_ptt

Resequence Level:component

Resequence Mode:Standard

Resequence Options

Group:<<Group Expression>>

Start:1

Timeout:0

ID:<<ID Expression>>

Increment:1

The options in the **Resequence Options** section change depending on the resequencing mode you select.

22.3.2 How to Configure the Resequencing Strategy

This section provides instructions on how to configure the three different types of resequencing strategies.

To configure a standard resequencer:

- 1. Set the resequence level as described in [Section 22.3.1, "How to Specify the Resequencing Level"](#).
- 2. On the Mediator Editor under either the Oracle Mediator component or the operation you want to configure, select **Standard** from the **Resequence Mode** dropdown list.

The Resequence Options box appears and includes the options for the standard resequencer, as shown in [Figure 22–4](#).

Figure 22–4 Oracle Mediator with Resequence Mode set to Standard

Resequence Mode:Standard

Resequence Options

Group:<<Group Expression>>

Start:1

Timeout:

ID:<<ID Expression>>

Increment:1

- 3. Fill in the fields listed in [Table 22–6](#).

Note:

You can either enter the XPath expressions directly in the Group and ID fields or you can click Invoke Expression Builder to the right of each field. This launches the Expression Builder, which provides graphical assistance in creating field expressions and adding functions.

Table 22–6 Standard Resequencing Options

Field Name	Description	Default Value	Mandatory
Group	The XPath that points to the field in the incoming message on which grouping is done. If you are editing the MPLAN file directly, the corresponding element is named <code>groupIDExpression</code> .	N/A	N
ID	The XPath that points to the field in the incoming message on which resequencing is done. If you are editing the MPLAN file directly, the corresponding element is named <code>sequenceIDExpression</code> .	N/A	Y
Timeout	The time period in seconds to wait for an expected message. The resequencer locks the group as timed-out if a time out occurs. If you are editing the MPLAN file directly, the corresponding element is named <code>timeOutDuration</code> .	0 ¹	N
Start	The starting number of the ID sequence. If you are editing the MPLAN file directly, the corresponding element is named <code>sequenceStart</code> .	1	N
Increment	The increment of the ID sequence. If you are editing the MPLAN file directly, the corresponding element is named <code>sequenceIncrement</code> .	1	N

¹ This default value means that the timeout never happens for a group by default.

To configure a FIFO resequencer:

1. Set the resequence level as described in [Section 22.3.1, "How to Specify the Resequencing Level"](#).
2. On the Mediator Editor under either the Oracle Mediator component or the operation you want to configure, select **FIFO** from the **Resequence Mode** dropdown list.

The Resequence Options box appears and includes the option for the standard resequencer, as shown in [Table 22–5](#).

Figure 22–5 Oracle Mediator with Resequence Mode set to FIFO

The screenshot shows the 'Resequence Mode' dropdown menu set to 'FIFO'. To its right is the 'Resequence Options' section, which contains a 'Group:' label followed by a text input field containing '<<Group Expression>>' and a search icon.

3. In the **Group** field, enter the XPath expression pointing to the field in the incoming message on which grouping is performed.

Notes: If you are modifying the MPLAN file directly, the name of the XML element is `groupIDExpression`. There is no default value, and the field is not mandatory.

To configure a best effort resequencer:

1. Set the resequence level as described in [Section 22.3.1, "How to Specify the Resequencing Level"](#).
2. On the Mediator Editor under either the Oracle Mediator component or the operation you want to configure, select **Best Effort** from the **Resequence Mode** dropdown list.

The Resequence Options box appears and includes the option for the standard resequencer, as shown in [Figure 22–6](#).

Figure 22–6 Oracle Mediator with Resequence Mode set to Best Effort

3. Fill in the fields listed in [Table 22–7](#) to configure the best effort resequencer.

Tip: You can specify either a maximum number of rows to process at one time or a time window for the messages. You cannot specify both.

4. If needed, you can change the percent of the time window that is added as a buffer. You configure the buffer using the Oracle Enterprise Manager Fusion Middleware Control.

For instructions, see “Configuring Resequenced Messages” in the *Oracle Fusion Middleware Administrator’s Guide for Oracle SOA Suite and Oracle BPM Suite*.

Table 22–7 Best Effort Resequencing Options

Field Name	Description	Default Value	Mandatory
Group	The XPath that points to the field in the incoming message on which grouping is performed. If you are editing the MPLAN file directly, the corresponding element is named <code>groupIDExpression</code> .	N/A	N
ID	The XPath that points to the field in the incoming message that contains the ID on which resequencing is performed. If you are editing the MPLAN file directly, the corresponding element is named <code>sequenceIDExpression</code> .	N/A	Y
Datatype	The data type of the sequence ID. The ordering process is based on the data type. Supported values are datetime and numeric. If you are editing the MPLAN file directly, the corresponding element is named <code>sequenceIDDataType</code> .	Numeric	Y

Table 22–7 (Cont.) Best Effort Resequencing Options

Field Name	Description	Default Value	Mandatory
Time Window	The length of time in minutes to wait after a message arrives to select messages from the data store for resequencing. You must specify a time window or the maximum rows (described below), but not both. If you are editing the MPLAN file directly, the corresponding element is named <code>timeWindow</code> .	0	N
Max Rows	Number of in-sequence messages that the resequencer should pick from the data store at a time. If you are editing the MPLAN file directly, the corresponding element is named <code>maxRowsRetrieved</code> .	5	N

22.4 Limitations in the Resequencer

The following limitation of resequencer has been noted in this release:

Resequencer Fails If XSD File Contains Multibyte Characters That the Server Locale Encoding Does Not Support

If the XSD file contains multibyte characters that the server locale encoding does not support, then the resequencer execution fails after triggering the project flow.

Understanding Message Exchange Patterns of an Oracle Mediator

This chapter describes common message exchange patterns between an Oracle Mediator service component and other applications.

This chapter includes the following sections:

- [Section 23.1, "Understanding a One-way Message Exchange Pattern"](#)
- [Section 23.2, "Understanding a Request-Reply Message Exchange Pattern"](#)
- [Section 23.3, "Understanding a Request-Reply-Fault Message Exchange Pattern"](#)
- [Section 23.4, "Understanding a Request-Callback Message Exchange Pattern"](#)
- [Section 23.5, "Understanding a Request-Reply-Callback Message Exchange Pattern"](#)
- [Section 23.6, "Understanding a Request-Reply-Fault-Callback Message Exchange Pattern"](#)

Notes: The following exchange patterns show the default handling of responses, faults, and callbacks by Oracle JDeveloper when a routing rule is created. Keep in mind the following points for all the cases:

- When a response, fault, or callback is sent back to the caller, it is also possible to route the same message to a different target service or event by clicking the button next to the target and selecting a different target.
 - When the caller of the Oracle Mediator expects a response, one or more routing rules may route the request to a target that does not return a response, but there should be *at least one* sequential routing rule that returns a response.
 - If you have multiple routing rules involved in a request-response interaction, then the routing rules that send the response back to the initial caller should precede other routing rules, if any, that forward the response.
 - The asynchronous request-reply pattern in Oracle Mediator is supported only for web services. This exchange pattern is not supported for adapters and other services.
-

23.1 Understanding a One-way Message Exchange Pattern

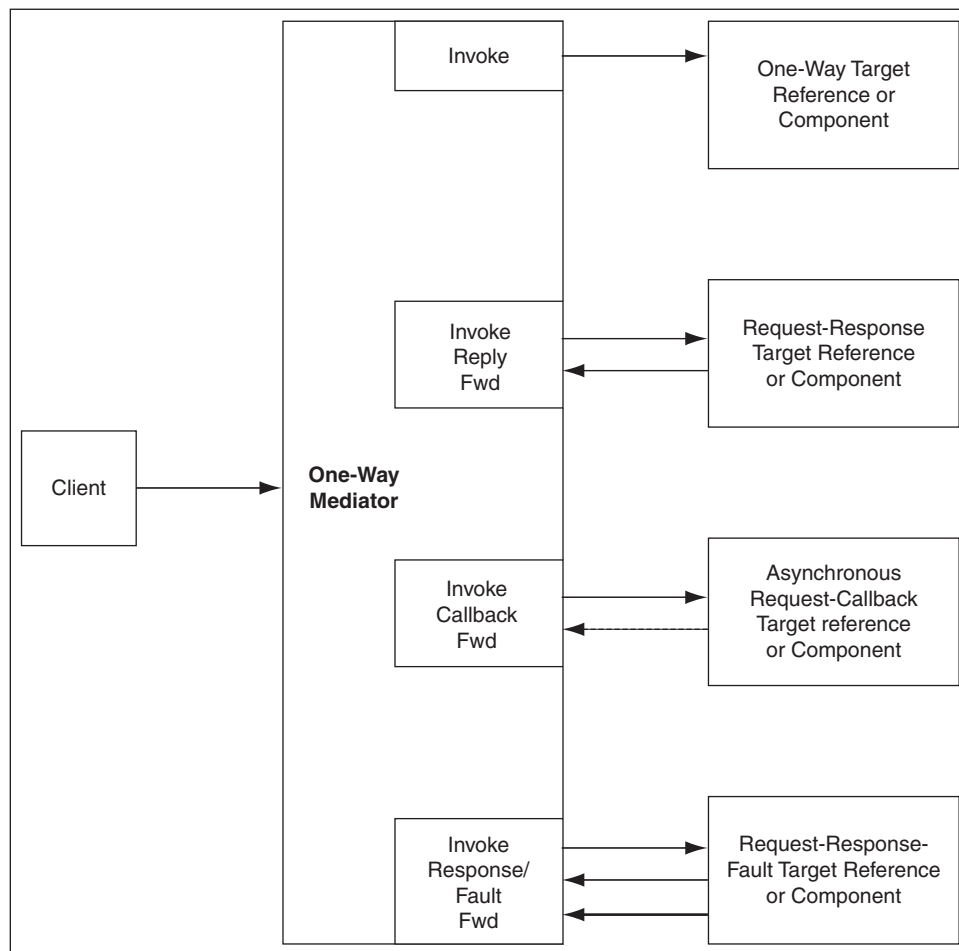
In a one-way interaction, the Oracle Mediator is invoked, but it does not send a response back to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 23-1](#):

Table 23-1 *Response When Oracle Mediator's WSDL Is a One-way Interaction*

Routing Rule Target Type	Response
Request	No response.
Request Response	Response is forwarded to another target or event.
Request Response Fault	Response and fault are forwarded to another target or event.
Request Callback	Callback is forwarded to another target or event.
Request Response Callback	Response and callback are forwarded to another target or event.
Request Response Fault Callback	Response, fault, and callback are forwarded to another target or event.

[Figure 23-1](#) illustrates the one-way message exchange pattern.

Figure 23-1 *One-way Message Exchange Pattern*



23.1.1 The one.way.returns.fault Property

The `one.way.returns.fault` property controls how faults and one-way messages are handled for one-way interface SOAP calls. You can add this property to the service binding component of the web service section for one-way web services in the `composite.xml` file. This property is *not* applicable to references. It is applicable only to services and only to the `binding.ws` binding type. [Table 23–2](#) provides more details on this property.

Table 23–2 *one.way.returns.fault* Property

If <code>one.way.returns.fault</code> Is...	Then...
Set to true: <pre> . . . <service name="Mediator1_2" ui:wsdlLocation="ReadFile.wsdl"> <interface.wsdl interface="http://xmlns.oracle.com/pcbpel/adapter/file /LocalSandbox/Project1/ReadFile%2F#wsdl.interface(Read_ ptt)"/> <binding.ws port="http://xmlns.oracle.com/pcbpel/adapter/file /LocalSandbox/Project1/ReadFile%2F#wsdl.endpoint (Mediator1/Read_pt)"> <property name="one.way.returns.fault" type="xs:string" many="false" override="may">true</property> </binding.ws> </service> . . . </pre>	Any fault that occurs during downstream processing returns a SOAP fault to the client and an HTTP response code of 500. (The same behavior as 11g Release 1.)
Set to false: <pre> . . . <service name="Mediator1_2" ui:wsdlLocation="ReadFile.wsdl"> <interface.wsdl interface="http://xmlns.oracle.com/pcbpel/adapter/file/ Local Sandbox/Project1/ReadFile%2F#wsdl.interface(Read_ ptt)"/> <binding.ws port="http://xmlns.oracle.com/pcbpel/adapter/file/LocalSan dbox/Project1/ReadFile%2F#wsdl.endpoint (Mediator1/Read_ pt)"> <property name="one.way.returns.fault" type="xs:string" many="false" override="may">>false</property> </binding.ws> </service> . . . </pre>	Any fault that occurs during downstream processing returns only an HTTP response code of 500. No SOAP fault is returned to the client.
Not set (the default case)	Any fault that occurs during downstream processing returns a SOAP fault to the client and an HTTP response code of 500. (The same behavior as 11g Release 1.)

To add the `one.way.returns.fault` property:

1. In the SOA Composite Editor, select the service binding component to which you want to add the `one.way.returns.fault` property.
2. Go to the Property Inspector section in the lower right part of the editor.
3. In the **Binding Properties** section, click the **Add** icon.
The Create Property dialog is displayed.
4. In the **Name** field, enter `one.way.returns.fault`.
5. In the **Value** field, enter `true` or `false`.
6. Click **OK**.

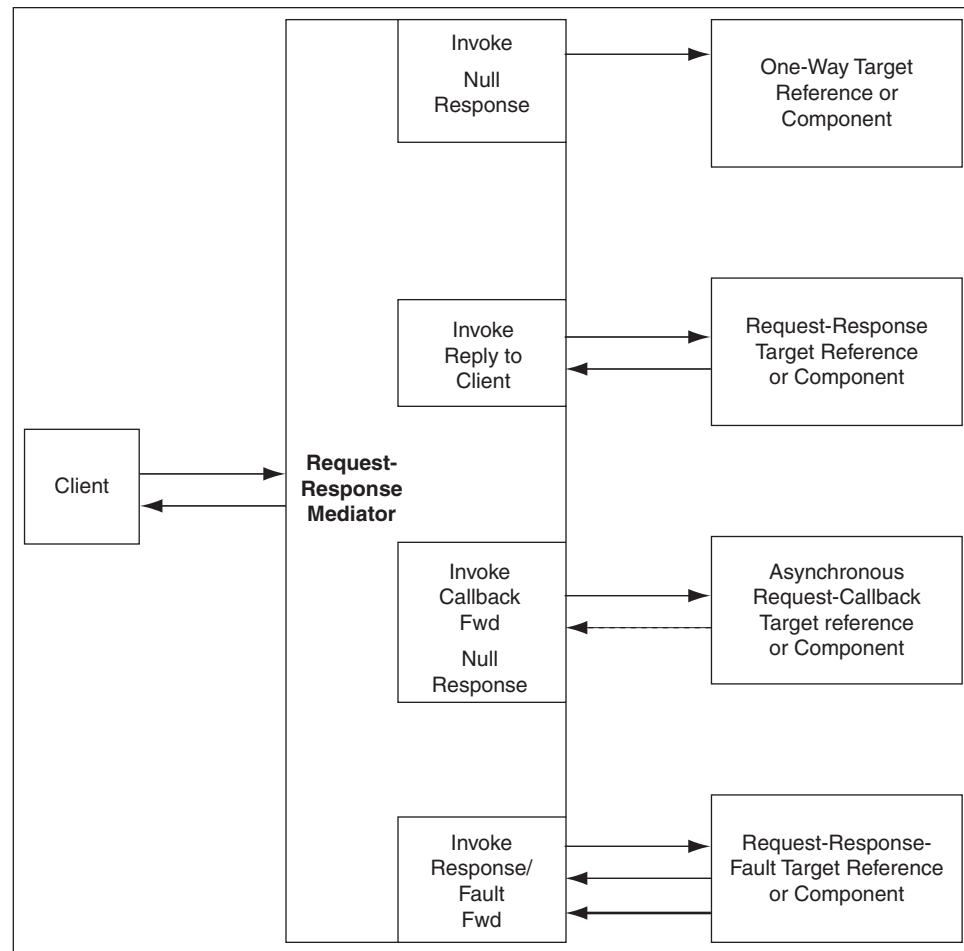
23.2 Understanding a Request-Reply Message Exchange Pattern

In a request-reply interaction, the Oracle Mediator is invoked and sends a reply to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 23–3](#):

Table 23–3 *Response When Oracle Mediator's WSDL Is a Request Reply*

Routing Rule Target Type	Response
Request	There is no response from the target, but there should be at least one sequential routing rule with a request-response service.
Request Response	The response is sent back to the caller. The response can be forwarded to another target or event, but there should be at least one sequential routing rule that returns a response back to the caller.
Request Response Fault	The response is sent back to the caller. The fault is forwarded to another target or event.
Request Callback	There is no response from the target, but there should be at least one sequential routing rule with a request-response service. The callback is forwarded to another target or event.
Request Response Callback	The response is sent back to the caller. The callback is forwarded to another target or event.
Request Response Fault Callback	The response is sent back to the caller. The callback and fault are forwarded to another target or event.

[Figure 23–2](#) illustrates the request-reply message exchange pattern.

Figure 23–2 Request-Reply Message Exchange Pattern

23.3 Understanding a Request-Reply-Fault Message Exchange Pattern

In a request-reply-fault interaction, the Oracle Mediator is invoked and sends a reply and one or more faults back to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 23–4](#):

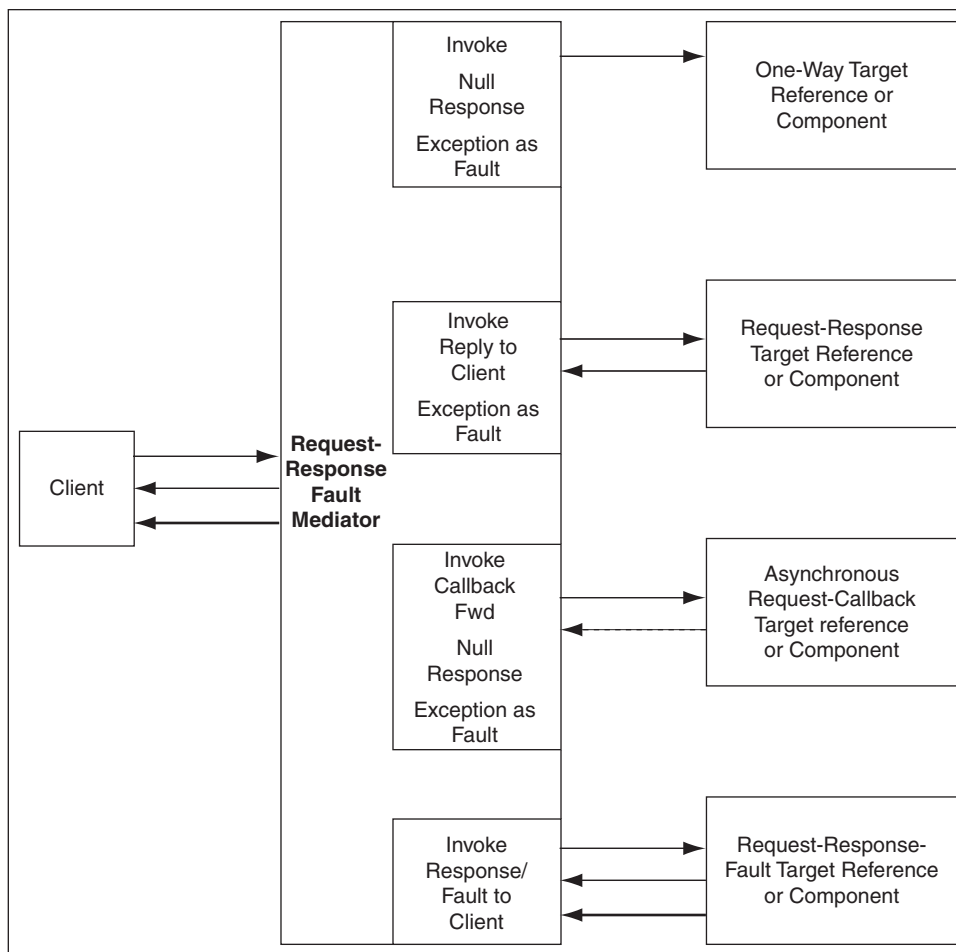
Table 23–4 Response When Oracle Mediator's WSDL Is a Request Reply Fault

Routing Rule Target Type	Response
Request	There should be at least one sequential routing rule with a request-response-fault service. Oracle Mediator returns <code>null</code> when there is no response to be sent.
Request Response	The response is sent back to the caller. Any exception in Oracle Mediator message processing may result in a fault.
Request Response Fault	The response and fault are sent back to the caller. Any exception in Oracle Mediator message processing may result in a fault.
Request Callback	There is no response from the target, but there should be at least one sequential routing rule with a request-response service. Oracle Mediator returns <code>null</code> when there is no response to be sent. The callback is forwarded to another target or event.
Request Response Callback	The response is sent back to the caller. Any exception in Oracle Mediator message processing may result in a fault.

Table 23–4 (Cont.) Response When Oracle Mediator's WSDL Is a Request Reply Fault

Routing Rule Target Type	Response
Request Response Fault Callback	The response and fault are sent back to the caller. Any exception in Oracle Mediator message processing may result in a fault.

Figure 23–3 illustrates the request-reply-fault message exchange pattern.

Figure 23–3 Request-Reply-Fault Message Exchange Pattern

23.4 Understanding a Request-Callback Message Exchange Pattern

In a request-callback interaction, the Oracle Mediator is invoked and may send an asynchronous reply to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 23–5:

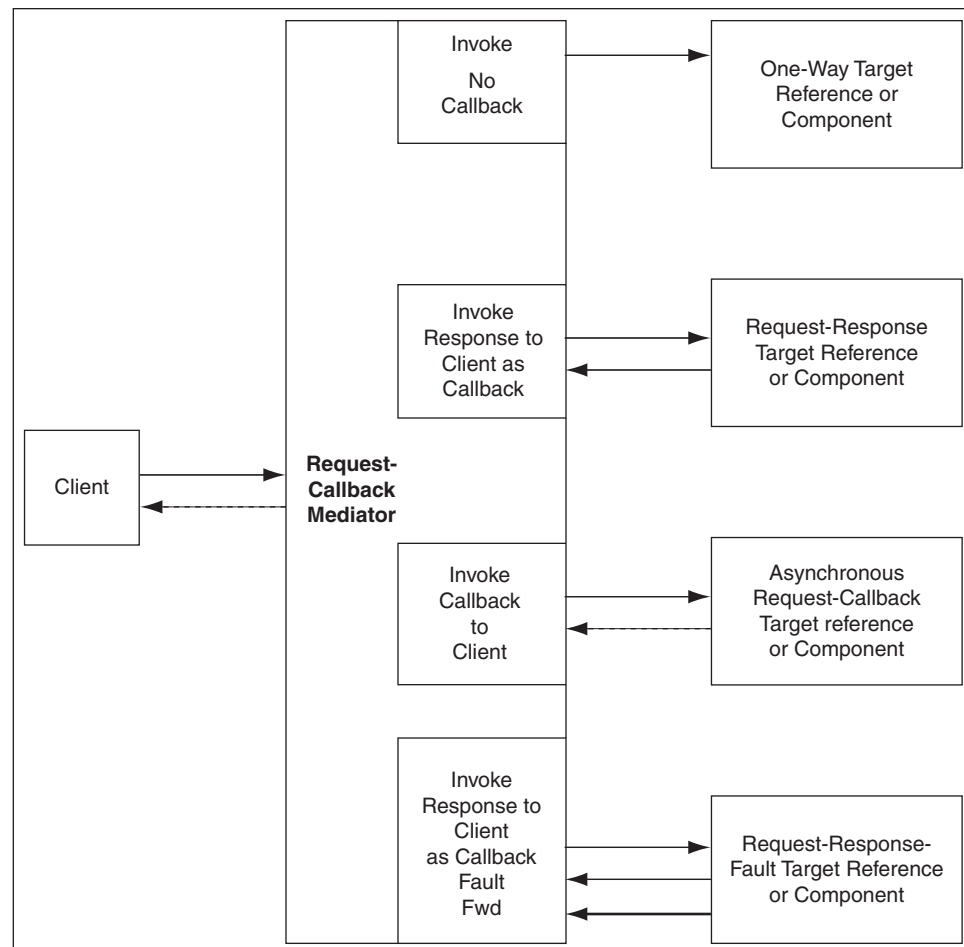
Table 23–5 Response When Oracle Mediator's WSDL Is a Request Callback

WSDL of the Routing Rule Target	Response
Request	There should be at least one sequential routing rule with a request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.

Table 23–5 (Cont.) Response When Oracle Mediator's WSDL Is a Request Callback

WSDL of the Routing Rule Target	Response
Request Response	The response is sent back to the caller, as a callback, in a separate thread.
Request Response Fault	The response is sent back to the caller, as a callback, in a separate thread. The fault is forwarded to another target or event.
Request Callback	The callback is sent back to the caller.
Request Response Callback	The callback is sent back to the caller, and the response is forwarded to another target or event.
Request Response Fault Callback	The callback is sent back to the caller. The response and fault are forwarded to another target or event.

Figure 23–4 illustrates the request-callback message exchange pattern.

Figure 23–4 Request-Callback Message Exchange Pattern

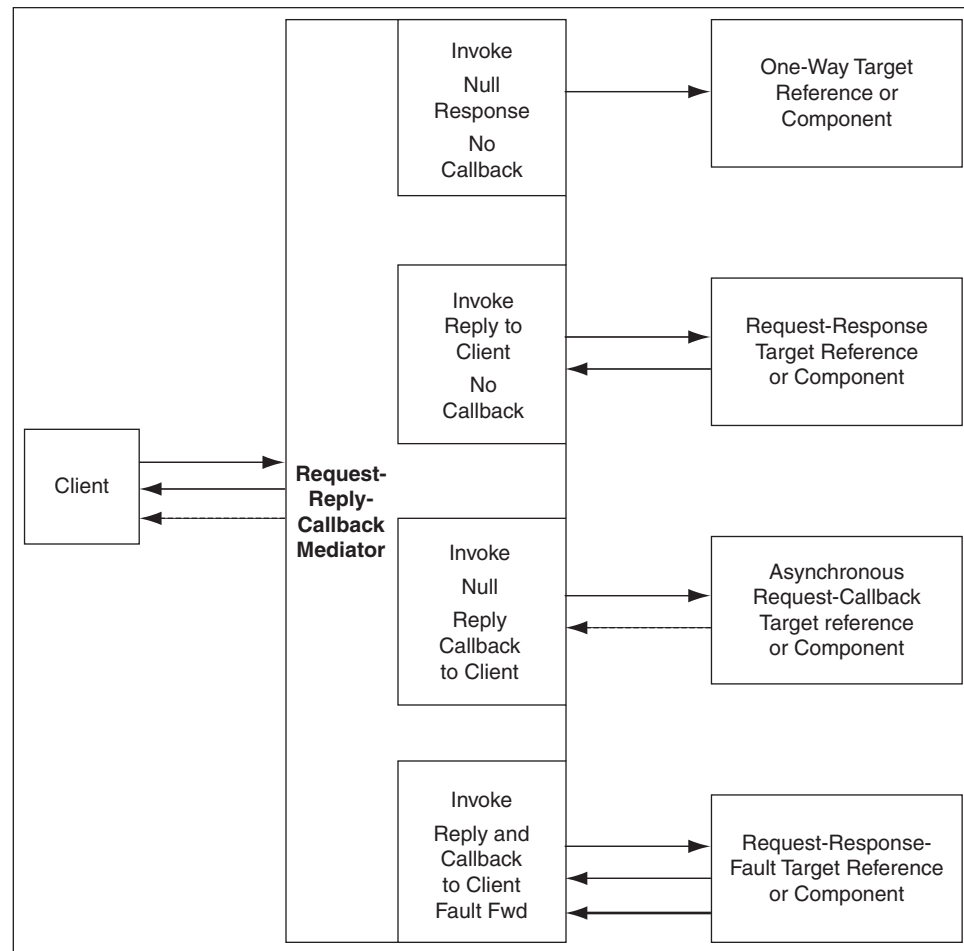
23.5 Understanding a Request-Reply-Callback Message Exchange Pattern

In a request-reply-callback interaction, the Oracle Mediator is invoked and sends a response and an asynchronous reply to the initial caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 23–6](#):

Table 23–6 *Response When Oracle Mediator's WSDL Is a Request Response Callback*

Routing Rule Target Type	Response
Request	There should be at least one sequential routing rule that returns a response. No callback is sent to the caller if there is no routing rule with a defined callback.
Request Response	There should be at least one sequential routing rule that returns a response. No callback is sent if there is no routing rule with a defined callback.
Request Response Fault	There should be at least one sequential routing rule that returns a response. No callback is sent to the caller if there is no routing rule with a defined callback.
Request Callback	There should be at least one sequential routing rule that returns a response. Oracle Mediator returns <code>null</code> when there is no response to be sent.
Request Response Callback	The response and callback are sent back to the caller.
Request Response Fault Callback	The response and callback are sent back to the caller. The fault is forwarded to another target or event.

[Figure 23–5](#) illustrates the request-reply-callback message exchange pattern.

Figure 23–5 Request-Reply-Callback Message Exchange Pattern

23.6 Understanding a Request-Reply-Fault-Callback Message Exchange Pattern

In a request-reply-fault-callback interaction, the Oracle Mediator is invoked and sends a response, an asynchronous reply, and one or more fault types to the initial caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 23–7](#):

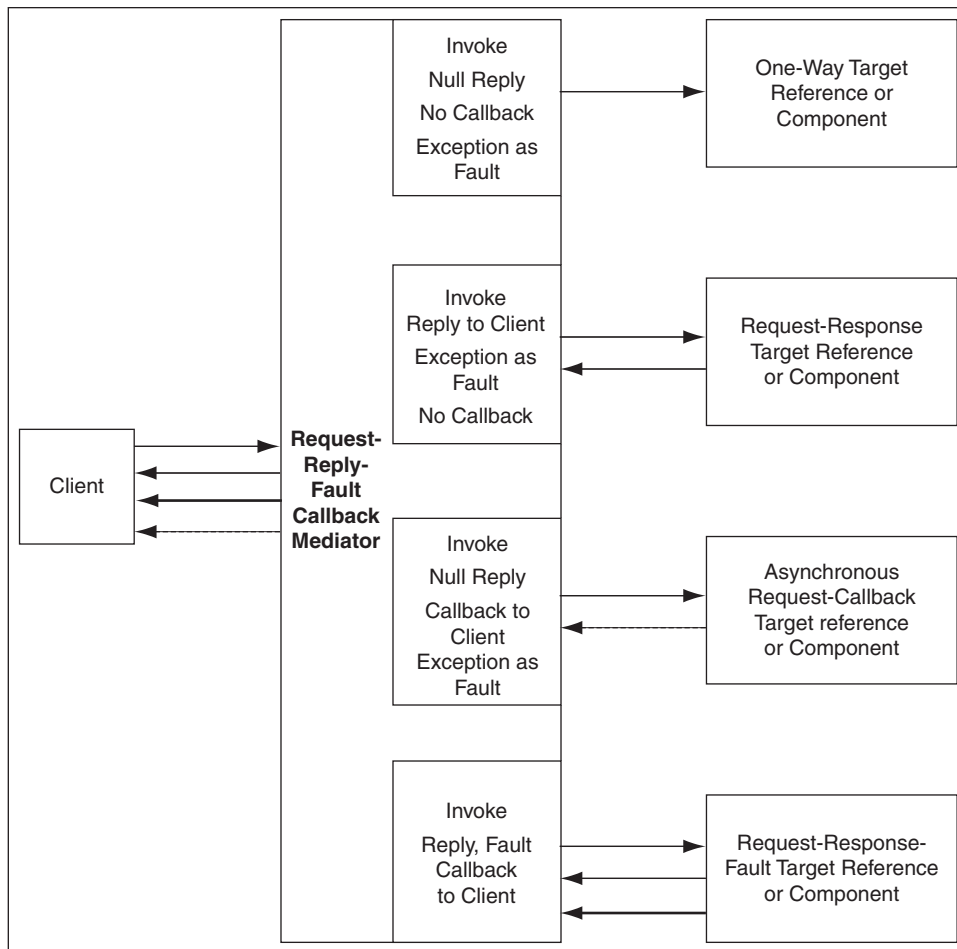
Table 23–7 Response to a Request Response Fault Callback Oracle Mediator

WSDL of the Routing Rule Target	Response
Request	There should be at least one sequential routing rule with a request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.
Request Response	There should be at least one sequential routing rule with a request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.
Request Response Fault	There should be at least one sequential routing rule with a request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.

Table 23–7 (Cont.) Response to a Request Response Fault Callback Oracle Mediator

WSDL of the Routing Rule Target	Response
Request Callback	There should be at least one sequential routing rule that returns a response. Oracle Mediator returns null when there is no response to be sent.
Request Response Callback	The response and callback are sent back to the caller. Any exception in Oracle Mediator message processing may result in a fault.
Request Response Fault Callback	The response, fault, and callback are sent back to the caller.

Figure 23–6 illustrates the request-reply-fault-callback message exchange pattern.

Figure 23–6 Request-Reply-Fault-Callback Message Exchange Pattern

Part IV

Using the Business Rules Service Component

This part describes how to use the business rules service component.

This part contains the following chapters:

- [Chapter 24, "Getting Started with Oracle Business Rules"](#)
- [Chapter 25, "Using Declarative Components and Task Flows"](#)

Getting Started with Oracle Business Rules

This chapter describes how to use a business rule service component to integrate a SOA composite application with Oracle Business Rules. A business rule service component is also called a Decision component. You can add business rules as part of an SCA application or as part of a BPEL process.

This chapter includes the following sections:

- [Section 24.1, "Introduction to the Business Rule Service Component"](#)
- [Section 24.2, "Overview of Rules Designer Editor Environment"](#)
- [Section 24.3, "Introduction to Creating and Editing Business Rules"](#)
- [Section 24.4, "Adding Business Rules to a BPEL Process"](#)
- [Section 24.5, "Adding Business Rules to a SOA Composite Application"](#)
- [Section 24.6, "Running Business Rules in a Composite Application"](#)
- [Section 24.7, "Using Business Rules with Oracle ADF Business Components Fact Types"](#)

For more examples of using Oracle Business Rules, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

24.1 Introduction to the Business Rule Service Component

A Decision component, also called a business rule service component, supports use of Oracle Business Rules in a SOA composite application. Decision components support the following SOA composite usage:

- A Decision component can be used within a SOA composite and wired to a BPEL component.
- A Decision component can be used within a SOA composite and used directly to run business rules.
- A Decision component can be used with the dynamic routing capability of Mediator.

For more information, see [Chapter 19, "Creating Oracle Mediator Routing Rules."](#)

- A Decision component can be used with the Advanced Routing Rules in Human Workflow.

For more information, see [Section 27.4, "Associating the Human Task Service Component with a BPEL Process."](#)

24.1.1 Integrating BPEL Processes, Business Rules, and Human Tasks

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 rules can be used include:

- **Dynamic processing**
Rules can perform intelligent routing within the business process based on service level agreements or other guidelines. For example, if the customer requires a response within one day, send a loan application to the QuickLoan loan agency only. If the customer can wait longer, then route the request to three different loan agencies.
- **Externalize business rules in the process**
There are typically many conditions that must be evaluated as part of a business process. However, the parameters to these conditions can be changed independently of the process. For example, you provide loans only to customers with a credit score of at least 650. This value may be changed dynamically based on new guidelines set by business analysts.
- **Data validation and constraint checks**
Rules can validate input documents or apply additional constraints on requests. For example, a new customer request must always be accompanied with an employment verification letter and bank account details.
- **Human task routing**
Rules are frequently used for human tasks in the business process:
 - Policy-based task assignments dispatch tasks to specific roles or users. For example, a process that handles incoming requests from a portal can route loan requests and insurance quotes to a different set of roles.
 - Load balancing of tasks among users. When a task is assigned to a set of users or a role, each user in that role acquires a set of tasks and acts on them in a specified time. For new incoming tasks, policies may be applied to set priorities on the task and put them in specific user queues. For example, a specific loan agent is assigned a maximum of 10 loans at any time.

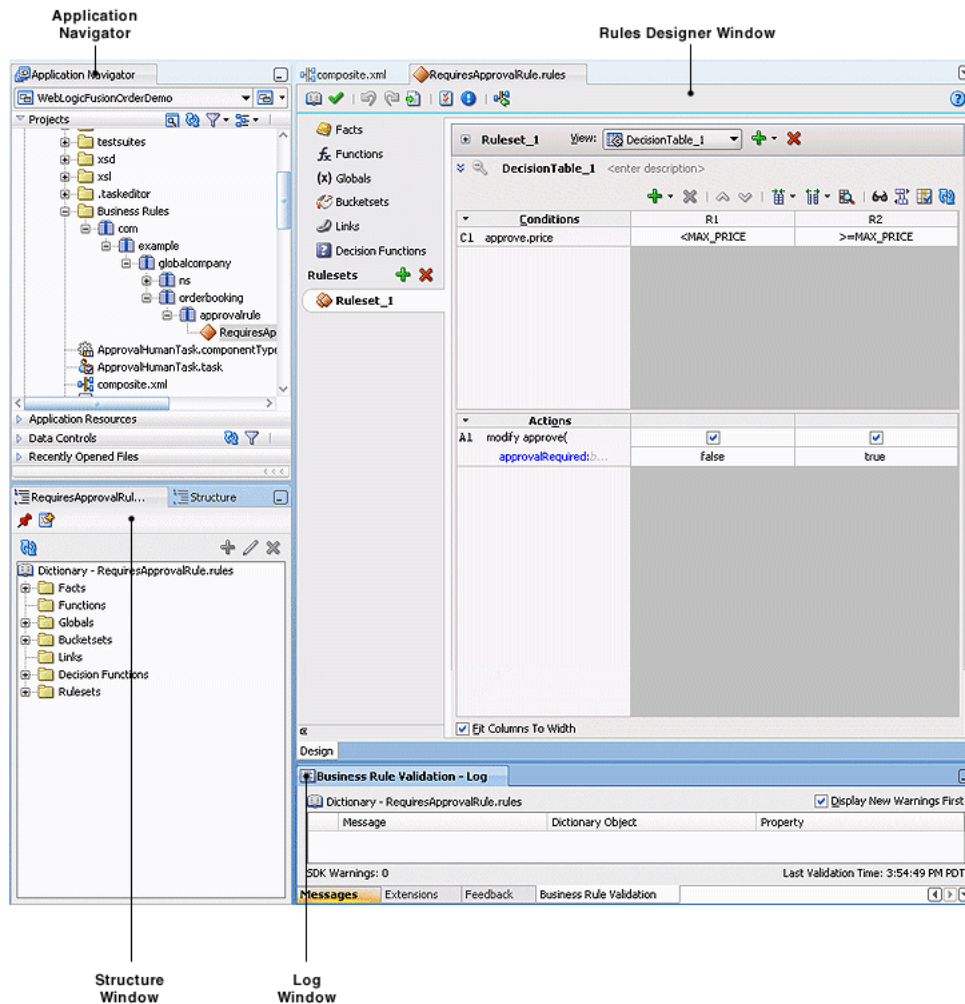
For more information about creating business rules in the Human Task editor of a human task component, see [Section 27.3.7.2, "Specifying Advanced Task Routing Using Business Rules."](#)

24.2 Overview of Rules Designer Editor Environment

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

The Business Rules Designer consists of the following main sections shown in [Figure 24–1](#). These sections enable you to work with business rules in Oracle JDeveloper.

Figure 24–1 Rules Designer in Oracle JDeveloper



24.2.1 Application Navigator

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

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

24.2.2 Rules Designer Window

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

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

- In a BPEL process, double click a business rule.
- In the Application Navigator, double-click a business rules dictionary file (a file with the **.rules** extension)
- Click the **Design** tab with a **.rules** file selected.

[Table 24–1](#) describes where you can find information about working with a dictionary with Rules Designer.

Table 24–1 Rules Designer Navigation Areas Descriptions

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

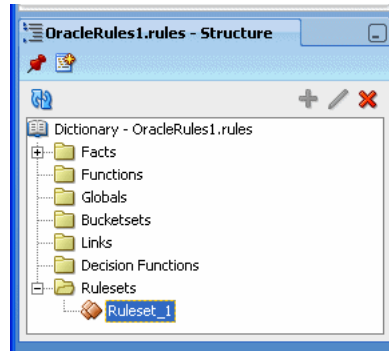
For more information about the Rules Designer navigation areas and its descriptions, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

24.2.3 Structure Window

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

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

[Figure 24–2](#) shows the Structure window.

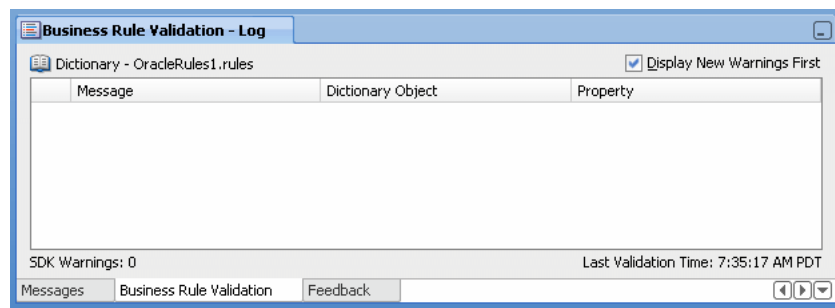
Figure 24–2 Structure Window with Rules Designer Dictionary

24.2.4 Business Rule Validation Log Window

Rules Designer displays the status of dictionary validation in the business rule validation log, as shown in [Figure 24–3](#).

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

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

Figure 24–3 Rules Designer Business Rule Validation Log

24.3 Introduction to Creating and Editing Business Rules

This section describes how to get started with business rules and provides a brief introduction to the main sections of Oracle JDeveloper that you use to design business rules.

24.3.1 How to Create Business Rules Components

You can add Business Rule components using the SOA Composite Editor.

To create a Business Rule component:

1. Follow the instructions in [Table 24–2](#) to start Oracle JDeveloper.

Table 24–2 Starting Oracle JDeveloper

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

2. Create a Business Rule service component through one of the following methods:
- As a service component in an existing SOA composite application:
- a. From the Component Palette, drag a **Business Rule** service component into the SOA Composite Editor.
- In a new application:
- a. From the Application Navigator, select **File > New > Applications > SOA Application**.
This starts the Create SOA Application wizard.
 - b. In the Name your application page, enter an application name in the **Name** field.
 - c. In the **Directory** field, enter a directory path in which to create the SOA composite application and project.
 - d. Click **Next**.
 - e. In the Name your project page, enter a unique project name in the **Project Name** field. The project name *must* be unique across SOA composite applications. This is because the uniqueness of a composite is determined by its project name. For example, do not perform the actions described in [Table 24–3](#).

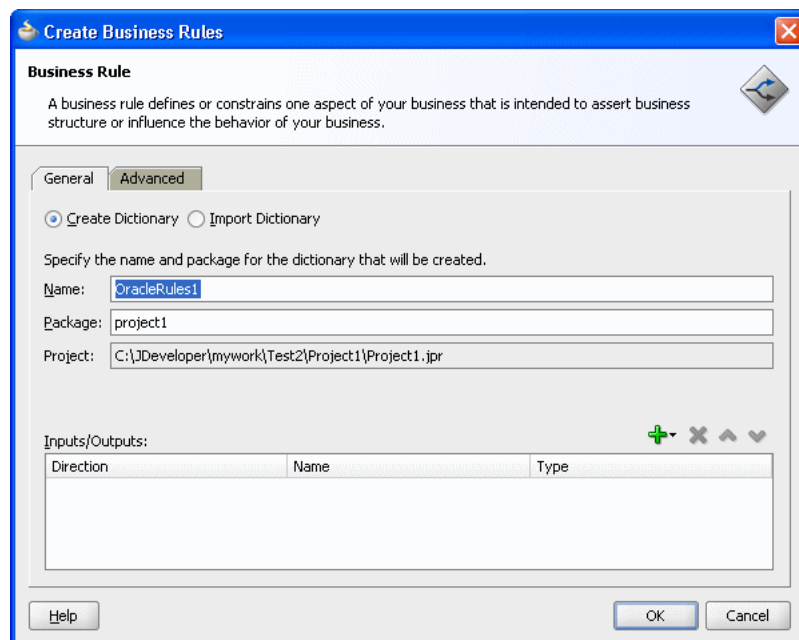
Table 24–3 Restrictions on Naming a SOA Project

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

During deployment, the second deployed project (composite) overwrites the first deployed project (composite).

- f. Click **Next**.
- g. In the Configure SOA settings page, select **Composite with Business Rule**.
- h. Click **Finish**.

Each method causes the Create Business Rules dialog shown in [Figure 24–4](#) to appear.

Figure 24–4 Create Business Rules Dialog

3. Provide the required details. For more information on providing Inputs and Outputs and on using the **Import Dictionary** option with this dialog, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.
4. Click **OK**.

24.3.2 Introduction to Working with Business Rules in Rules Designer

When you are working with business rules Oracle JDeveloper displays Rules Designer.

24.4 Adding Business Rules to a BPEL Process

You can use a Decision component, also called a business rule service component, to execute business rules in a BPEL process.

24.4.1 How to Add Business Rules to a BPEL Process

You add business rules to a BPEL process using a **Business Rule** component. When you add a business rule component to a BPEL process, you must include input and output variables to provide input to the rules and obtain results back from the business rules.

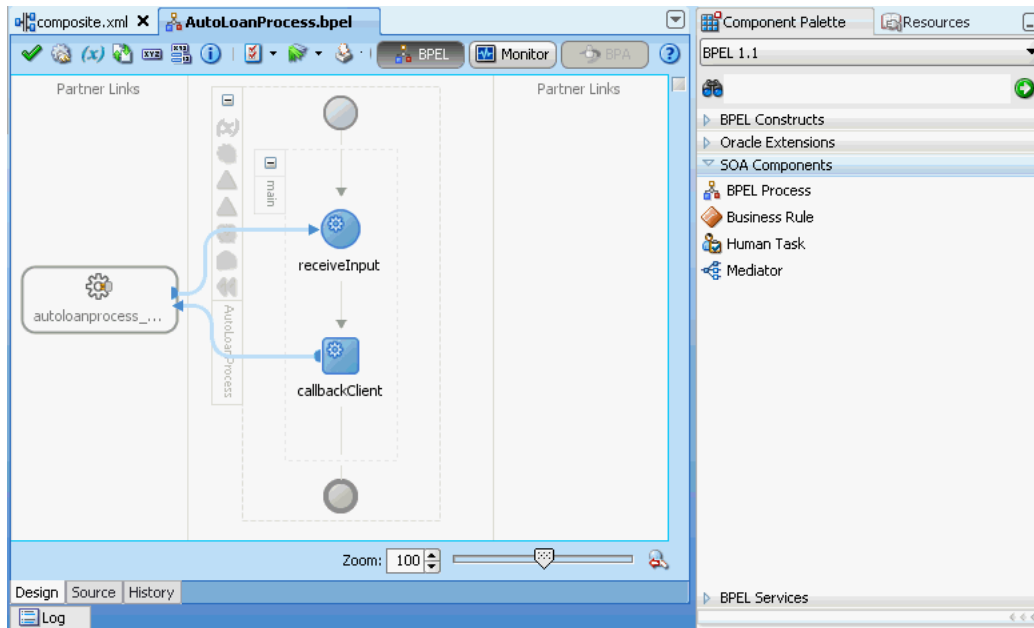
A business rule component enables you to execute business rules and make business decisions based on the rules. To create a business rule component, also called a Decision component, you drag-and-drop a **Business Rule** from the component palette into the BPEL process.

To add a business rule to a BPEL process:

1. Create a BPEL process service component. For more information, see [Section 4.1, "Introduction to the BPEL Process Service Component."](#)

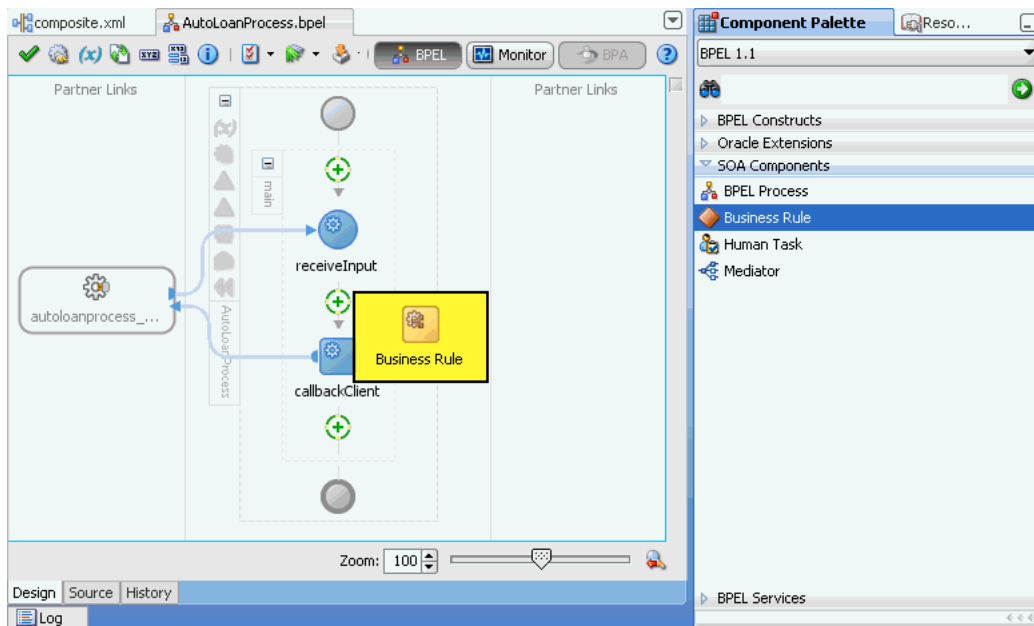
2. Expand the BPEL process by double-clicking the process item. For example, expand the BPEL process to view `receiveInput` and `callbackClient` as shown in [Figure 24–5](#).

Figure 24–5 Adding A Business Rule to a BPEL Process



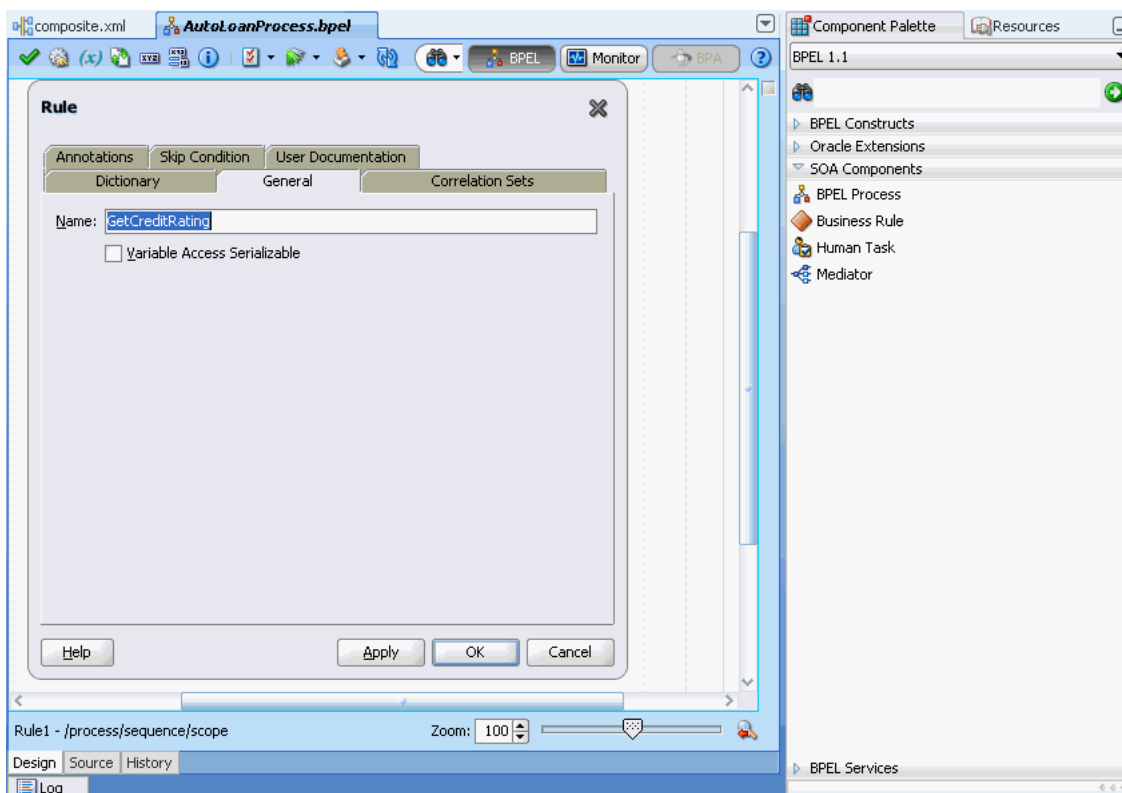
3. Select **Business Rule** from the SOA Components section of the Component Palette and drag-and-drop a **Business Rule** into the position where the business rules are needed. For example, drag-and-drop a **Business Rule** between `receiveInput` and `callbackClient`, as shown in [Figure 24–6](#).

Figure 24–6 Drag-and-drop a Business Rule into a BPEL Process



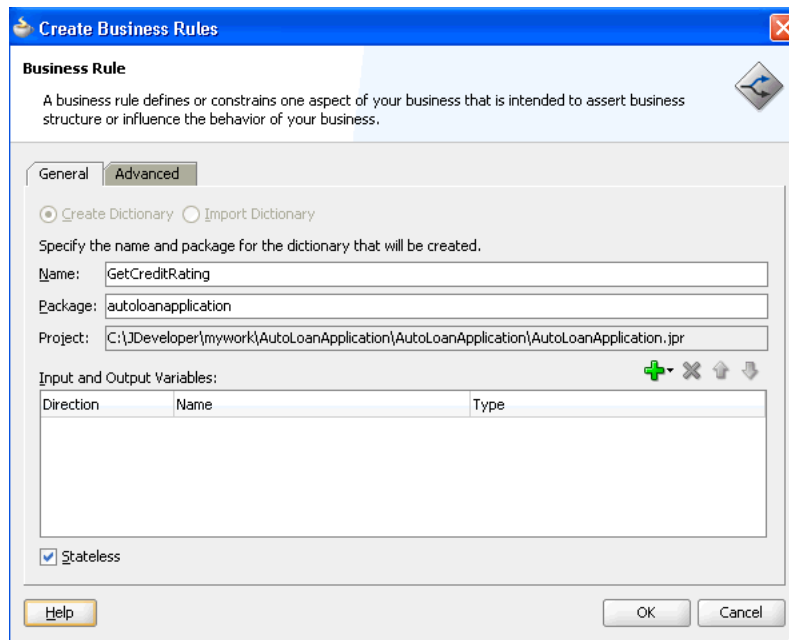
4. Oracle JDeveloper displays the business rule in the diagram. Double-click the business rule component to display the **Rule** dialog box. The **Rule** dialog box provides tabs, such as General, Dictionary, Correlation Sets, and so on, where you can select an existing Oracle Business Rules dictionary or enter the name of a new dictionary. Under the **General** tab, in the **Name** field enter a name for the business rule. For example, enter `GetCreditRating`, as shown in [Figure 24-7](#). If you previously created a dictionary, under the **Dictionary** tab, in the **Dictionary** field, select an existing dictionary.

Figure 24-7 Business Rule Added to Auto Loan BPEL Process



5. In the Business Rule area for the **Business Rule Dictionary**, click the **Create Dictionary** icon to display the Create Business Rules dialog.
6. In the Create Business Rules dialog you do the following:
 - Specify a name for the Oracle Business Rules dictionary and a package name.
 - Specify the input and output data elements for the business rule. For example, for a sample Decision component named `GetCreditRating`, the input is a rating request document. The output is generated when you run the business rules, and for this example is a rating document. For example, in BPEL you can create two new variables, `RatingRequest` and `Rating` that carry the input and output data for the `GetCreditRating` rules.

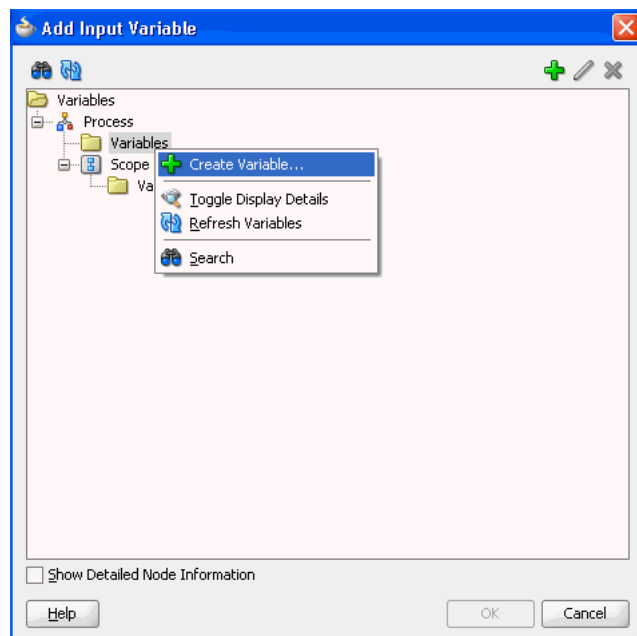
Enter a name for the Oracle Business Rules dictionary. For example, enter `GetCreditRating`, as shown in [Figure 24-8](#).

Figure 24–8 Adding GetCreditRating Business Rule Dictionary**Add inputs for business rule:**

1. In the Create Business Rules dialog, from the menu next to the **Add** icon select **Add Input Variable...** to create the input variable.

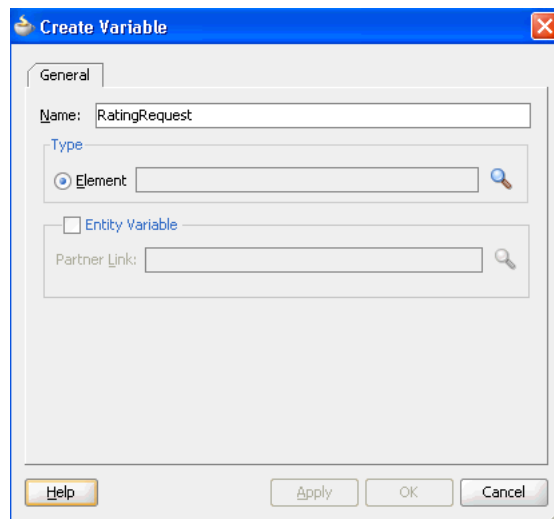
This displays the **Add Input Variable** dialog box.

2. In the **Add Input Variable** dialog box, expand the **Process** folder and select the **Variables** folder immediately inside the **Process**.
3. Right-click the **Variables** folder, and from the list select **Create Variable...** as shown in [Figure 24–9](#).

Figure 24–9 Add Input Variable

This displays the **Create Variable** dialog box.

4. In the **Create Variable** dialog box, in the **Name** field enter a value. For example, enter `RatingRequest` as shown in [Figure 24–10](#).

Figure 24–10 Create Variable Dialog

5. In the Create Variable **Type** area click the **Browse Elements** icon. Use the navigator to locate the schema element type for the input variable. For example, select the `ratingrequest` type. Add any needed types using the Type Chooser.
6. Click the **Import Schema File** icon to import the schema. For example, import `CreditRatingTypes.xsd`. Also import any other required schema for your application.
7. In the Type Chooser dialog, select **ratingrequest** and click **OK**.
8. In the Create Variable dialog, click **OK**.

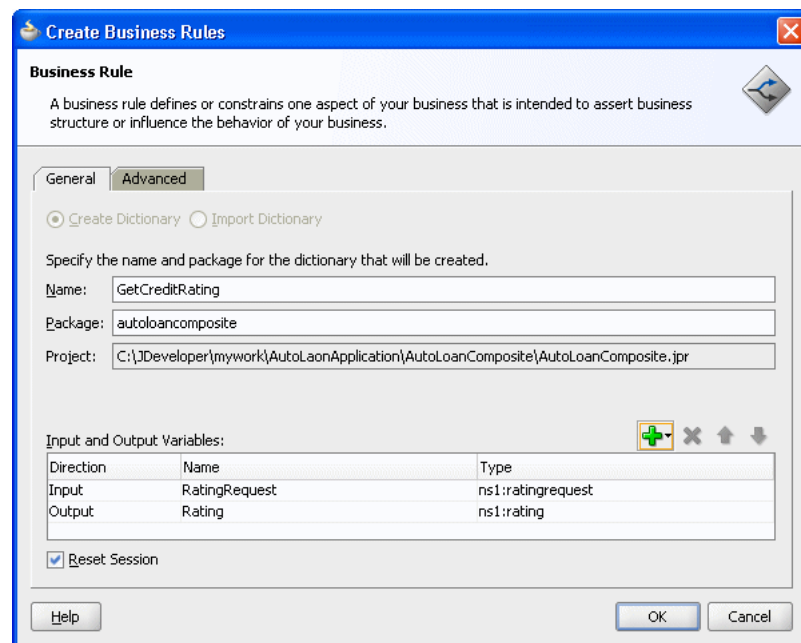
9. In the Add Input Variable dialog, click **OK**.

Add outputs for business rule:

1. In the Create Business Rules dialog, from the dropdown menu next to the **Add** icon, select **Add Output Variable....** This displays the Add Output Variable dialog. Use this dialog to create an output variable. For example, create an output variable for `GetCreditRating` in the same way you created the input variable.
2. In the Add Output Variable dialog select the scope by selecting the **Variables** folder under **Process**.
3. Right-click and from the dropdown list select **Create Variable....** This displays the Create Variable dialog.
4. In the Create Variable dialog, in the **Name** field enter the output variable name. For example enter `Rating`.
5. In the Create Variable dialog, in the **Type** area select the **Browse elements** icon and use the Type Chooser dialog to enter the type for the output variable. For example, expand the `CreditRatingTypes.xsd` and select the element type `rating`.
6. In the Type Chooser dialog, click **OK**.
7. In the Create Variable dialog, click **OK**.
8. In the Add Output Variable dialog, click **OK**.

This displays the Create Business Rules dialog, as shown in [Figure 24–11](#).

Figure 24–11 Create Business Rules Dialog with Input and Output Variables

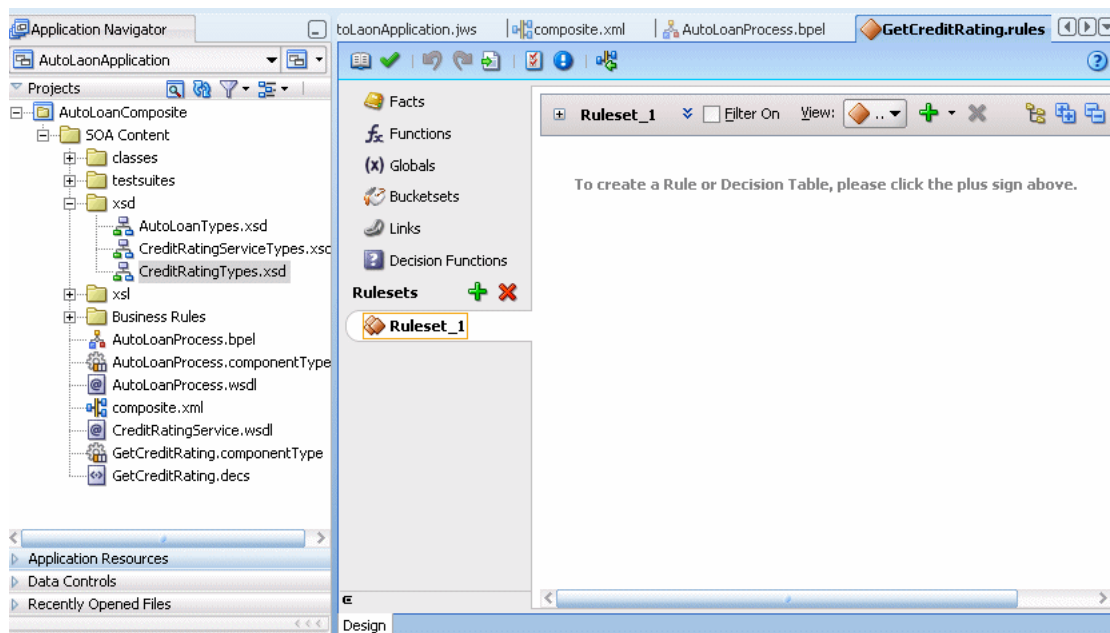


Set options and create decision service and business rules dictionary:

1. If you do not want to use the default service name, then select the **Advanced** tab and in the **Service Name** field enter the service name. For example enter the service name `CreditRatingService`.

2. Determine if the Decision Component is stateful or stateless with **Reset Session**. For more information, see [Section 24.4.5, "What You May Need to Know About Decision Component Stateful Operation"](#).
3. In the Create Business Rules dialog, click **OK**. Oracle JDeveloper creates the Decision component and the dictionary and displays Rules Designer, as shown in [Figure 24-12](#).

Figure 24-12 Rules Designer Canvas Where You Work with Business Rules



For information on Rules Designer, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

24.4.2 What Happens When You Add Business Rules to a BPEL Process

When you add business rules to a BPEL process, Oracle JDeveloper creates a Decision component to control and run the business rules using the Business Rule Service Engine.

A Decision component consists of the following:

- Rules or Decision Tables that are evaluated using the Rules Engine. These are defined using Rules Designer and stored in a business rules dictionary.
- A description of the facts required for specific rules to be evaluated and the decision function to call. Each ruleset that contains rules or Decision Tables is exposed as a service with facts that are input and output, and the name of an Oracle Business Rules decision function. The facts are exposed through XSD definitions when you define the inputs and outputs for the business rule. A Decision function is stored in an Oracle Business Rules dictionary. For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.
- A web service wraps the input, output, and the call to the underlying Business Rule service engine.

This web service lets business processes assert and retract facts as part of the process. In some cases, all facts can be asserted from the business process as one

unit. In other cases, the business process can incrementally assert facts and eventually consult the rule engine for inferences. Therefore, the service supports both stateless and stateful interactions.

You can create a variety of such Decision components.

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

24.4.3 What Happens When You Create a Business Rules Dictionary

After you create an application, a project, and a rules dictionary, the rules dictionary appears in the structure pane in Oracle JDeveloper and Rules Designer opens in the main canvas.

As part of the create Business Rule dialog you either select an existing dictionary or a new rule dictionary is created with the following pre-loaded data:

- XML fact type model based on the input and output information of the Business Rule.
- A Ruleset that must be completed by adding rules or Decision Tables. With an existing dictionary, you use the import option to specify a dictionary that may already contain the rules or Decision Tables.
- A service component with the input and output contract of the Decision component.
- A Decision component for the rule dictionary and wires to the BPEL process.

Note: When you create inputs and outputs for a business rule, the XML fact type that is created in the associated dictionary is named based on the schema types for the inputs and outputs that you supply in the Create Business Rules dialog. When you specify schema type for the input and the output, Rules Designer defines fact types and aliases associated with your type selections for input and output. If you only use a single type for both the input and the output, then the Decision component creates a single fact that is shown in the Rules Designer Facts tab. This fact represents the fact type you specified and uses an alias name that is a concatenation of both the input variable name and the output variable name. In Rules Designer you can rename this alias if you do not like the default naming scheme for the fact type.

24.4.4 What You May Need to Know About Invoking Business Rules in a BPEL Process

When you add business rules to a BPEL process Oracle JDeveloper creates a Decision Service that supports calling Oracle Business Rules with the inputs you supply, and returning the outputs with results. The Decision Service provides access to Oracle Business Rules Engine at runtime as a web service. For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

24.4.5 What You May Need to Know About Decision Component Stateful Operation

A Decision Component running in a business rules service engine supports either stateful or stateless operation. The **Reset Session** checkbox in the Create Business Rules dialog provides support for these two modes of operation.

By default the **Reset Session** checkbox is selected which indicates stateless operation. Stateless operation means that, at runtime, the rule session is released after the Decision Component invocation.

When **Reset Session** is unselected, the underlying Oracle Business Rules object is kept in the memory of the business rules service engine at a separate location (so that it is not given back to the Rule Session Pool when the operation is finished). A subsequent use of the Decision component re-uses the cached RuleSession object, with all its state information from the `callFunctionStateful` invocation, and then releases it back to the Rule Session pool after the `callFunctionStateless` operation is finished. Thus, when **Reset Session** is unselected the rule session is saved for a subsequent request and a sequence of Decision Service invocations from the same BPEL process should always end with a stateless invocation.

24.5 Adding Business Rules to a SOA Composite Application

To work with Oracle Business Rules in a SOA composite application, you create an application and add business rules.

The business rule service component enables you to integrate your SOA composite application with business rules. This creates a business rule dictionary and enables you to execute business rules and make business decisions based on the rules.

After creating a project in Oracle JDeveloper, you must create a Business Rule Service component within the project. When you add a business rule you can create input and output variables to provide input to the service component and to obtain results from the service component.

To use business rules with Oracle JDeveloper, you do the following:

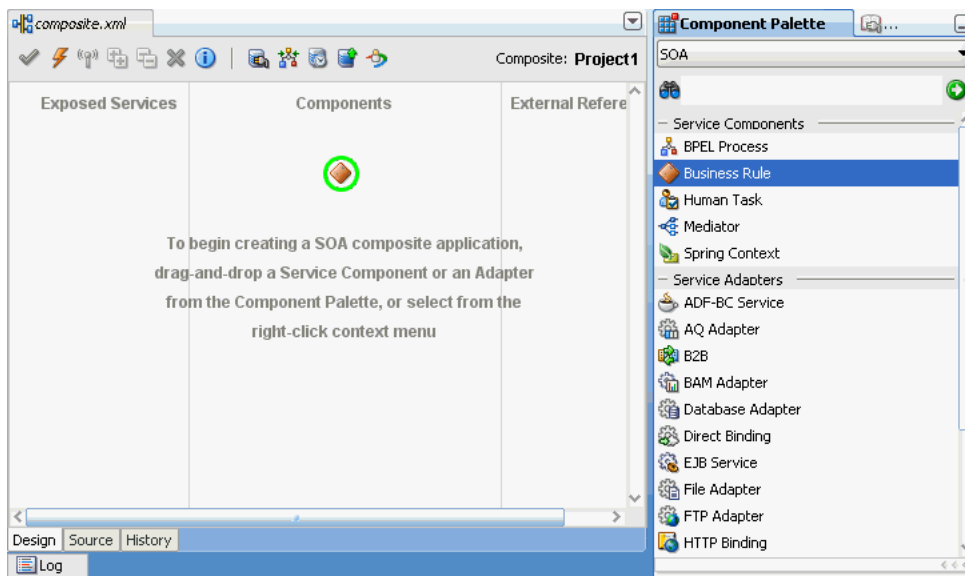
- Add a business rules service component
- Create input and output variables for the service component
- Create an Oracle Business Rules dictionary

24.5.1 How to Add Business Rules to a SOA Composite Application

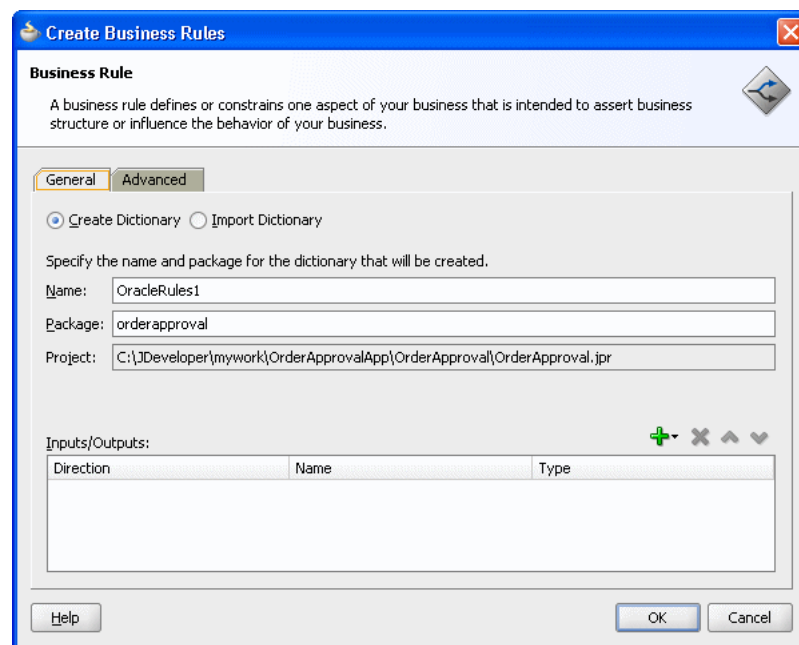
To work with Oracle Business Rules in a SOA composite application you use Oracle JDeveloper to create an application, a project, and then add a business rule component.

To create a SOA application with business rules:

1. Create a SOA application and project. For more information, see [Section 2.1.1, "How to Create a SOA Application and Project"](#). For a SOA composite using business rules, pick the required technologies for your application. For example, you may need the following for a SOA application with business rules: ADF Business Components, Java, and XML. You move these items to the **Selected** area on the Project Technologies tab.
2. In the Application Navigator, if the SOA composite editor is not showing, then in your project expand **SOA Content** folder and double-click `composite.xml` to launch the SOA composite editor.
3. From the Component Palette, drag-and-drop a **Business Rule** from the Service Components area of the SOA menu to the **Components** lane of the SOA composite editor, as shown in [Figure 24–13](#).

Figure 24–13 Adding Business Rules to a SOA Composite Application

4. When you drag-and-drop a **Business Rule**, Oracle JDeveloper displays the Create Business Rules dialog as shown in [Figure 24–14](#).

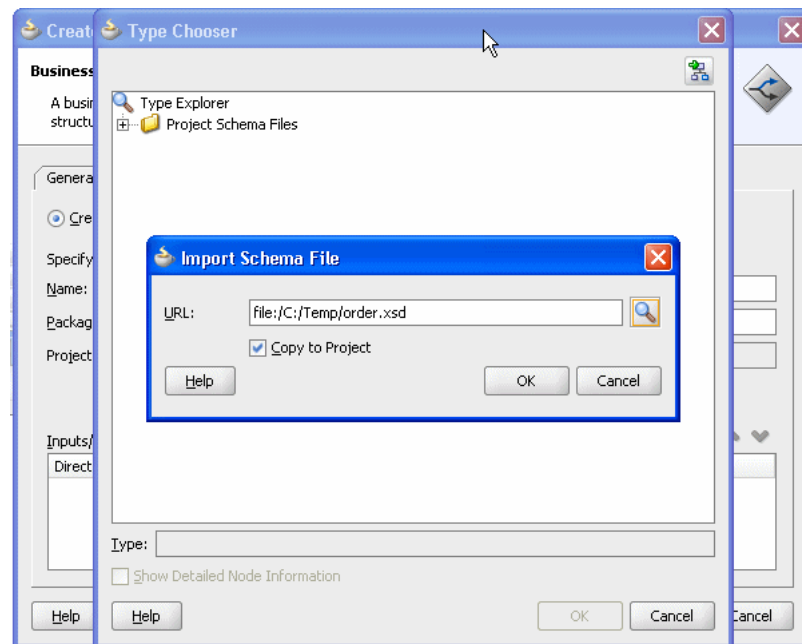
Figure 24–14 Adding Business Rules to a SOA Composite and Creating a Dictionary

Add inputs for business rules:

1. In the Create Business Rules dialog box, from the menu next to the **Add** icon select **Input...** to add input for the business rule. This displays the Type Chooser dialog.
2. In the Type Chooser dialog, add inputs. If the schema is available in the **Project Schema Files**, skip to step 9 to select the appropriate schema.
3. Click the **Import Schema File...** icon. This brings up the Import Schema File dialog.

4. In the Import Schema File dialog click **Browse Resources** to choose the XML schema elements for the input. This displays the SOA Resource Browser dialog.
5. In the SOA Resource Browser dialog, navigate to find the schema for your business rules input. For example, select the `order.xsd` schema file, and click **OK**.
6. In the Import Schema File dialog select **Copy to Project**, as shown in [Figure 24–15](#).

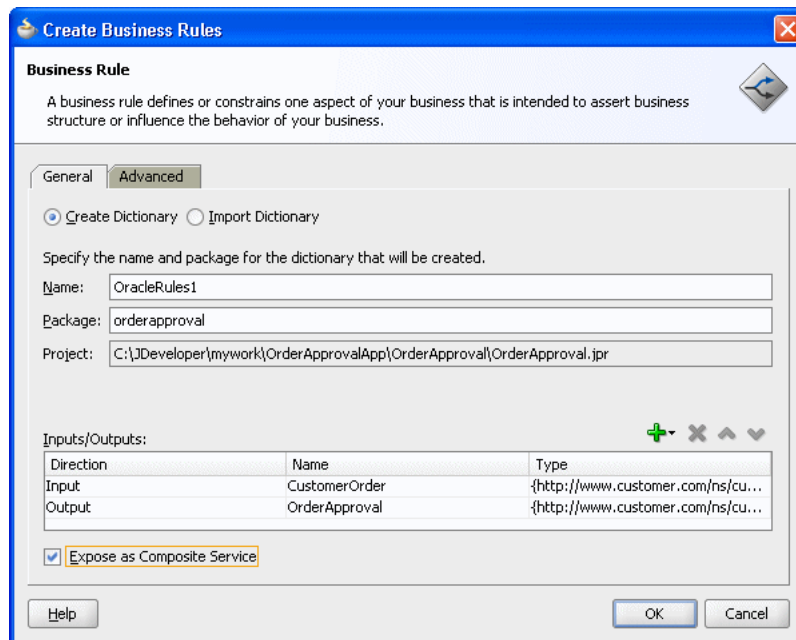
Figure 24–15 Importing Schema for Input to Business Rules



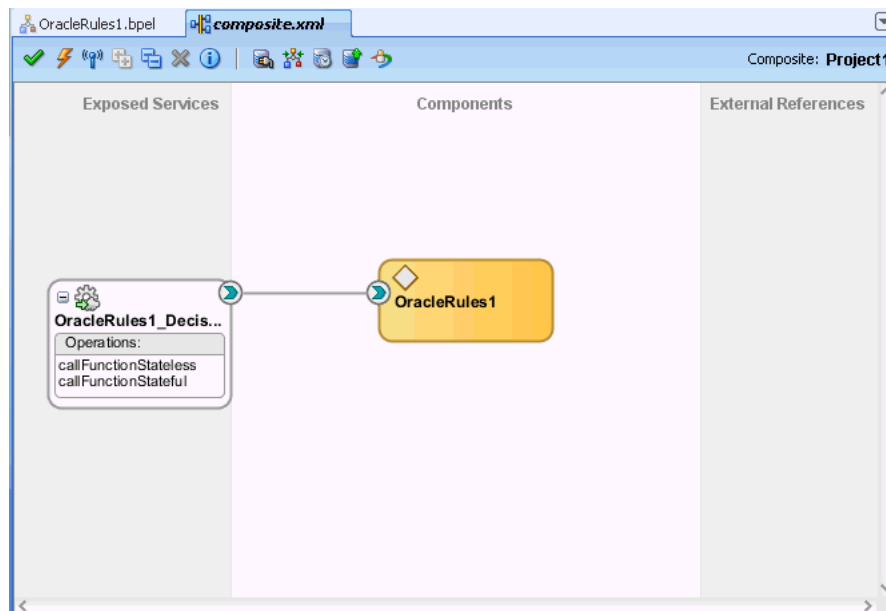
7. In the Import Schema File dialog, click **OK**.
8. In the Localize Files dialog, click **OK**.
9. Use the Type Chooser dialog navigator to locate and select the input from the schema and click **OK**. For example, select the `CustomerOrder` element as the input.

Add outputs for business rules:

1. In the Create Business Rules dialog, from the dropdown menu next to the **Add** icon select **Output....**
2. In the Type Chooser dialog, in a manner similar to adding an input add the output. For example, add `OrderApproval` from the `order.xsd` and click **OK**.
3. This displays the Create Business Rules dialog, as shown in [Figure 24–16](#).

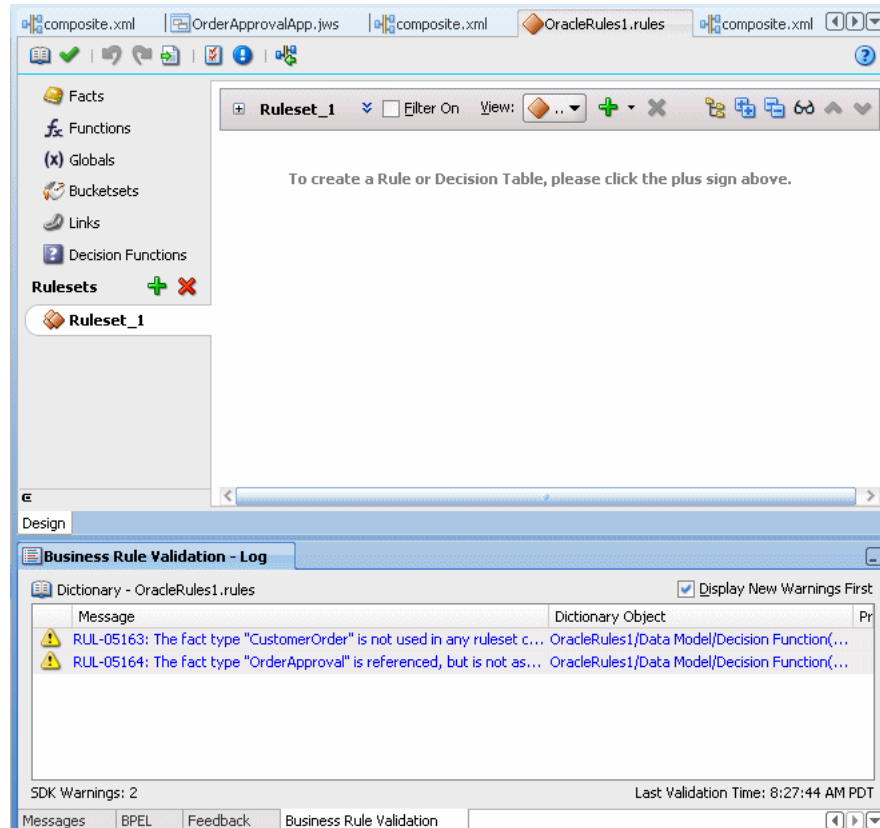
Figure 24–16 Create Business Rules Dialog with Input and Output**Set options and create decision service and business rules dictionary:**

1. In the Create Business Rules dialog, select **Expose as Composite Service**.
2. If you do not want to use the default service name, then select the **Advanced** tab and in the **Service Name** field enter the service name.
3. In the Create Business Rules dialog, click **OK**. This creates the Business Rule component, also called a Decision component, and Oracle JDeveloper shows the Business Rule component in the canvas workspace as shown in [Figure 24–17](#).

Figure 24–17 Business Rule Component in SOA Composite

4. Double-click the Decision component (for example the **OracleRules1** business rule). This opens Rules Designer, as shown in [Figure 24–18](#). The validation log shows validation warnings for the input and output facts. By working with Rules Designer to define rules or decision tables, you remove these warning messages.

Figure 24–18 Rules Designer Showing New Dictionary for SOA Composite Application



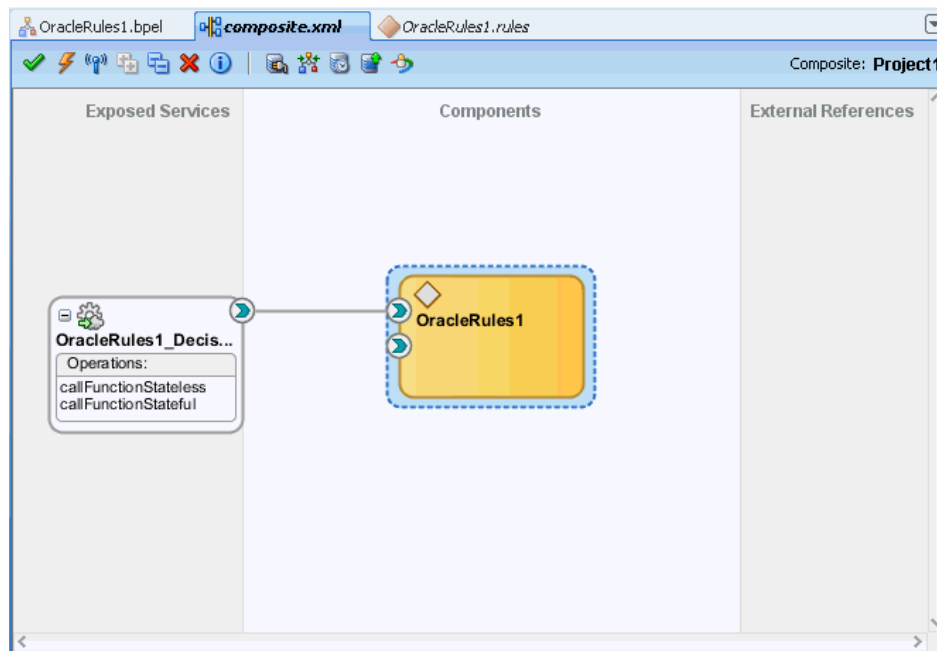
For information on Rules Designer, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

24.5.2 How to Select and Modify a Decision Function in a Business Rule Component

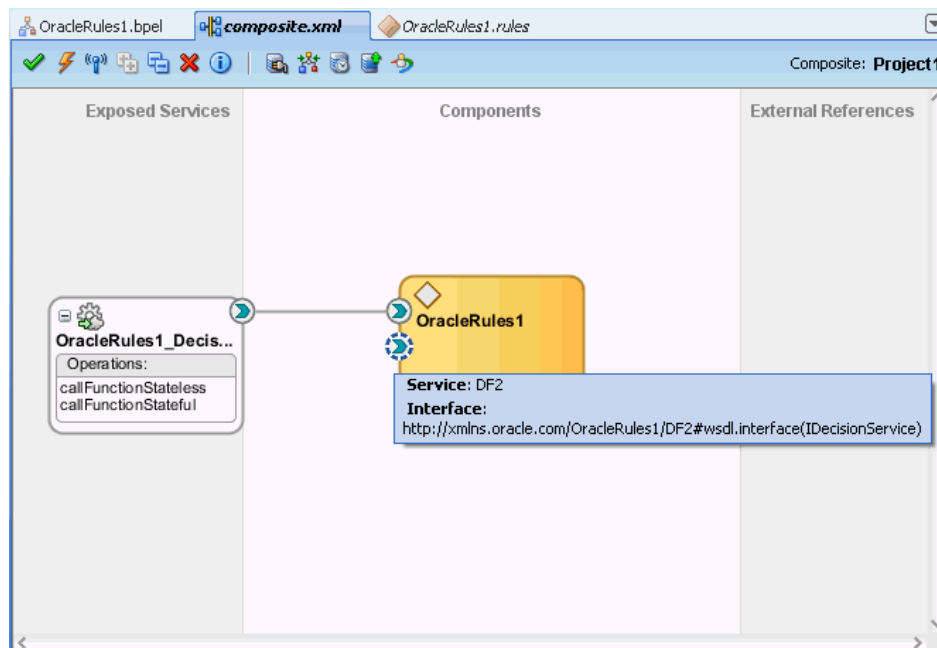
You can specify one or more decision functions as inputs for calling Oracle Business Rules as a component in a composite application. For example, you can specify a particular decision function as the input when multiple decision functions are available in an Oracle Business Rules dictionary.

To specify a decision function in a composite application:

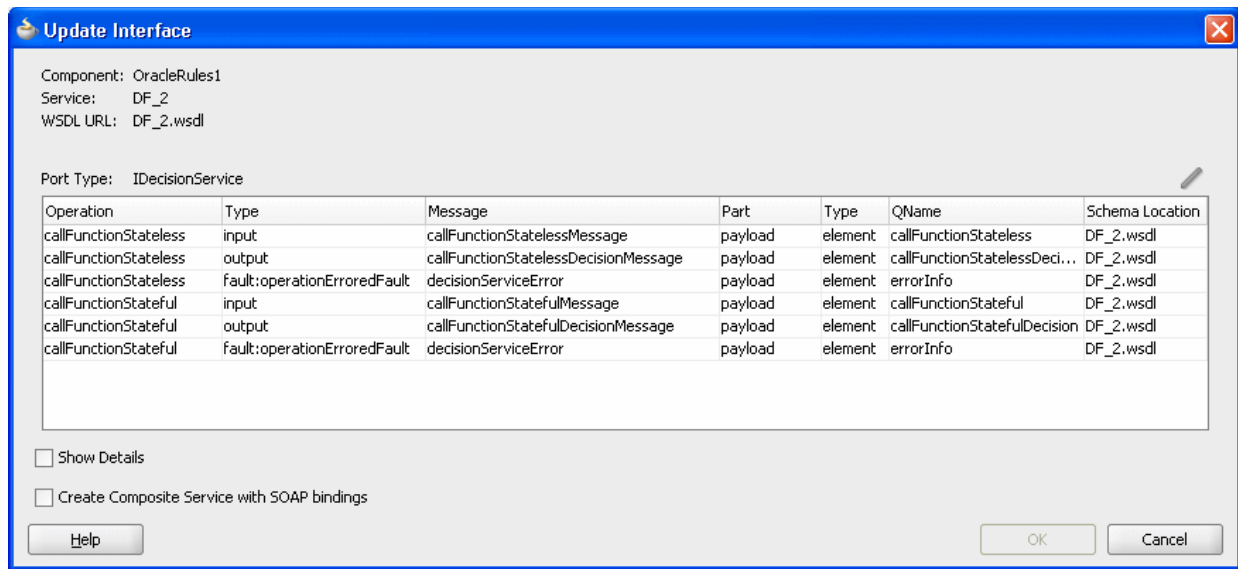
1. Add a decision function to the Oracle Business Rules dictionary. For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.
2. Add a Business Rule component to the composite application. For more information, see [Section 24.5.1, "How to Add Business Rules to a SOA Composite Application"](#).
3. Select a business rule component, as shown in [Figure 24–19](#).

Figure 24–19 Selecting a Business Rule Component in a Composite Application

4. Select the decision function port of interest. For example, select the port for DF_2 as shown in [Figure 24–20](#).

Figure 24–20 Selecting a Decision Function Port in a Business Rule Component

5. When you select the port, Oracle JDeveloper shows the port information in the Property Inspector.
6. When you double-click the port, Oracle JDeveloper displays the Update Interface dialog for the port as shown in [Figure 24–21](#).

Figure 24–21 Update Interface Dialog for a Decision Function in a Business Rule Decision Port

24.6 Running Business Rules in a Composite Application

You run business rules as part of a Decision component within a SOA composite application. The business rules are executed by the Business Rule Service Engine. You can use Oracle Enterprise Manager Fusion Middleware Control to monitor the Business Rule Service Engine and to test a SOA composite application that includes a Decision component. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

24.6.1 What You May Need to Know About Testing a Standalone Decision Service Component

To test a standalone Decision Service component by using Oracle Enterprise Manager Fusion Middleware Control, you must provide the name of the Decision Service as the value of the payload name field in the Test Web Service page as shown in [Figure 24–22](#).

Figure 24–22 Invoking a Standalone Test Decision Service

ORACLE Enterprise Manager 11g Fusion Middleware Control Setup Help Log Out

Farm SOA Infrastructure Topology

Rules [1.0123] Logged in as weblogic | Host: adc2190666.us.oracle.com

SOA Composite Page Refreshed Oct 2, 2010 9:20:08 AM PDT

Test Web Service

Use this page to test any WSDL, including WSDLs that are not in the farm. To test a Web service, enter the WSDL and click Parse WSDL. When the page refreshes with the WSDL details, first select the Service, then select the Port, and then select the Operation that you want to test. Specify any input parameters, and click Test Web Service.

WSDL Parse WSDL

HTTP Basic Auth Option for WSDL Access

Service OracleRules1_DecisionService_1_ep

Port IDecisionService_pt

Operation

Endpoint URL Edit Endpoint URL ☐

Request Response

- Security
- Quality of Service
- HTTP Transport Options
- Additional Test Options
- Input Arguments

Tree View

Name	Type	Value
* payload	payload	
@ * name	NCName	
configURL	string	
bpelInstance	tbpelProcess	
* parameterList	parameterList	

'name' in payload should be the Decision Service name as can be seen in the sample .decs file in [Figure 24–23](#).

Figure 24–23 Sample .decs File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<decisionServices xmlns="http://xmlns.oracle.com/bpel/rules"
  name="OracleRules1">
  <ruleEngineProvider name="OracleRulesSDK" provider="Oracle_11.0.0.0.0">
    <repository type="SCA-Archive">
      <path>Rules/oracle/rules/rules/OracleRules1.rules</path>
    </repository>
  </ruleEngineProvider>
  <decisionService targetNamespace="http://xmlns.oracle.com/OracleRules1/OracleRules1_DecisionService_1"
    ruleEngineProviderReference="OracleRulesSDK"
    name="OracleRules1_DecisionService_1">
    <catalog>OracleRules1</catalog>
    <pattern name="CallFunctionStateless">
      <arguments>
        <call>rules.OracleRules1.OracleRules1_DecisionService_1</call>
      </arguments>
    </pattern>
    <pattern name="CallFunctionStateful">
      <arguments>
        <call>rules.OracleRules1.OracleRules1_DecisionService_1</call>
      </arguments>
    </pattern>
  </decisionService>
</decisionServices>
```

Without the Decision Service name, it would not be possible to invoke the standalone Decision Service with just the payload and endpoint details.

24.7 Using Business Rules with Oracle ADF Business Components Fact Types

You can use Oracle ADF Business Components Fact Types and `ActionTypes` from the Business Rules Service Engine. Typically, a Decision component can be used within a SOA composite and wired to a BPEL component and the Oracle Business Rules rules act on XML types. The Business Rules Service Engine is called as a web service with a payload containing instances of the XML schema types, and the service engine returns a response similarly.

It is also possible to use Oracle ADF Business Components Fact Types from a Decision component. Instead of loading the Oracle ADF Business Components Fact Type instances and passing them to the Business Rules Service Engine, you call the Business Rules Service Engine with configuration information describing how the Oracle ADF Business Components view object rows can be loaded. Special Oracle Business Rules decision functions in the `DecisionPointDictionary` and classes in the Oracle Business Rules SDK Decision Point API then load the rows and assert Oracle ADF Business Components fact type instances. When working with Oracle ADF Business Components Fact Types, you write rules that use user-defined Java classes which inherit from `ActionType` to affect action, such as modifying the Oracle ADF Business Components fact type instances such that they update their underlying database rows.

A Decision component requires an XML document as input. The Oracle Business Rules Decision Point dictionary provides an XML Fact Type called `SimpleDecisionPointInput` that serves as this input. The primary key(s) of Oracle ADF Business Components are passed to the business rule service component. The business rule service component invokes a user-defined decision function which it invokes to load the Oracle ADF Business Components view object instances, asserts them in the rules engine, and then marshals the results in the following order:

1. `DecisionPointDictionary.DecisionPoint_Preprocessing_Webservice` Ruleset: The preprocessing ruleset reads the business component from the database and asserts them as facts.
2. User-defined rulesets: The user ruleset matches these facts and should assert facts that extend `ActionType` to update the business component.
3. `DecisionPointDictionary.DecisionPoint_Postprocessing_Webservice` Ruleset: The actual updating is performed by the postprocessing ruleset. Use of `ActionTypes` is optional.

For specific instructions on how to use Oracle ADF Business Components Fact Types and `ActionTypes` from the Business Rules Service Engine, see the source code for Oracle Business Rules-specific samples and SOA samples available online at

<https://soasamples.samplecode.oracle.com>

Using Declarative Components and Task Flows

This chapter describes how to use different Oracle Business Rules declarative components and task flows to develop high-performance, interactive, and multitiered applications that are also easy to maintain.

This chapter includes the following sections:

- [Section 25.1, "Introduction to Declarative Components and Task Flows"](#)
- [Section 25.2, "Using the Oracle Business Rules Editor Declarative Component"](#)
- [Section 25.3, "Using the Oracle Business Rules Dictionary Editor Declarative Component"](#)
- [Section 25.4, "Using the Oracle Business Rules Dictionary Task Flow"](#)
- [Section 25.5, "Localizing the ADF-Based Web Application"](#)

25.1 Introduction to Declarative Components and Task Flows

Declarative components are reusable, composite User Interface (UI) components that comprise other existing Application Development Framework (ADF) Faces components. Consider an application that contains multiple JSF pages. On a particular page, a set of specific components is used in multiple parts of that page. In this scenario, if you make any changes to any of the components in the set, you typically must replicate the changes in multiple parts of the page. This approach makes it difficult to maintain the consistency of the structure and layout of the page. However, by defining a declarative component that comprises the given set of components, you can reuse that composite declarative component in multiple places or pages. Declarative components, thereby, save time and ensure integrity across pages, because when you make any changes to the components, the JSF pages using them automatically get updated.

ADF task flows are reusable components that provide a modular and transactional method in specifying the control flow in an application. You can use a set of reusable task flows as an alternative to representing an application as a single large JSF page flow, thereby providing modularity. Each task flow contains a part of the entire navigational plan of the application. The nodes in a task flow are called activities. Apart from navigation, task flow activities can also call methods on managed beans or call another task flow without invoking any particular page. This facilitates reuse because business logic can be invoked independently of the page being displayed.

25.2 Using the Oracle Business Rules Editor Declarative Component

This section discusses the Oracle Business Rules Editor declarative component. It also provides information on how to create and run an application using the Rules Editor component, and then deploy the application. In addition, this section lists the supported tags and the localization process for the application.

25.2.1 Introduction to the Oracle Business Rules Editor Component

Oracle Business Rules Editor is a declarative component that can be embedded in any ADF-based Web application. The component renders the user interface for rules editing and handles all events associated with rules editing. Rules Editor uses the Rules SDK2 API to create and edit rules.

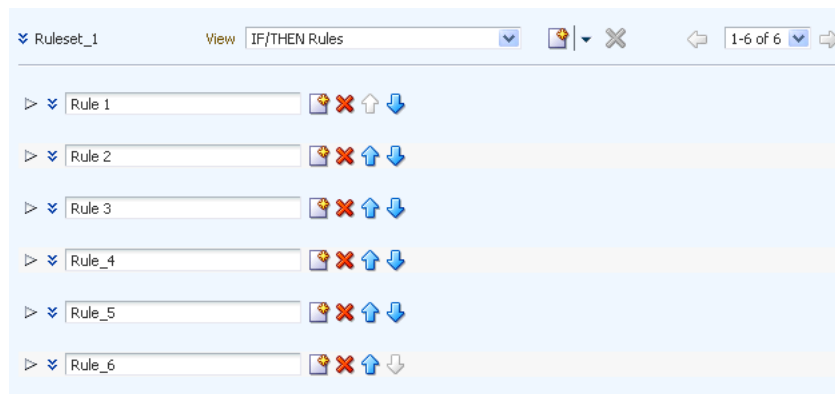
Note: You should not confuse Rules Editor with Rules Dictionary Editor. Rules Editor is used to edit rules inside a specified ruleset. In fact, Rules Editor is embedded within Rules Dictionary Editor. For more information about Rules Dictionary Editor, see [Section 25.3, "Using the Oracle Business Rules Dictionary Editor Declarative Component."](#)

Using Rules Editor, you can edit rules and decision tables that are part of a single ruleset. You require to specify a `RuleSetModel` object, which is a wrapper around the Rules SDK ruleset object, as a parameter to the Rules Editor component. If multiple rulesets are required to be modified, multiple Rules Editor components must be instantiated, one for each ruleset.

The Rules Editor component performs the following functions:

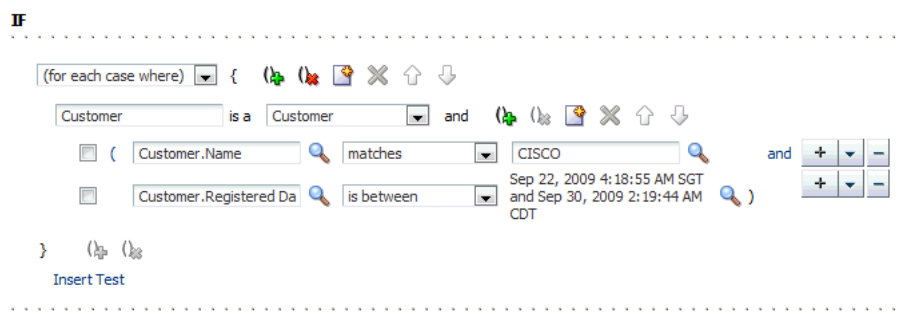
- Creates, updates, and deletes:
 - Rules in a ruleset, as shown in [Figure 25–1](#):

Figure 25–1 Rules in a Ruleset



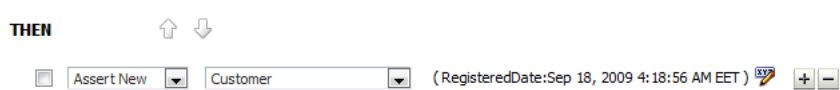
- Simple tests or conditions in a rule, as shown in [Figure 25–2](#):

Figure 25–2 Simple Tests or Conditions in a Rule



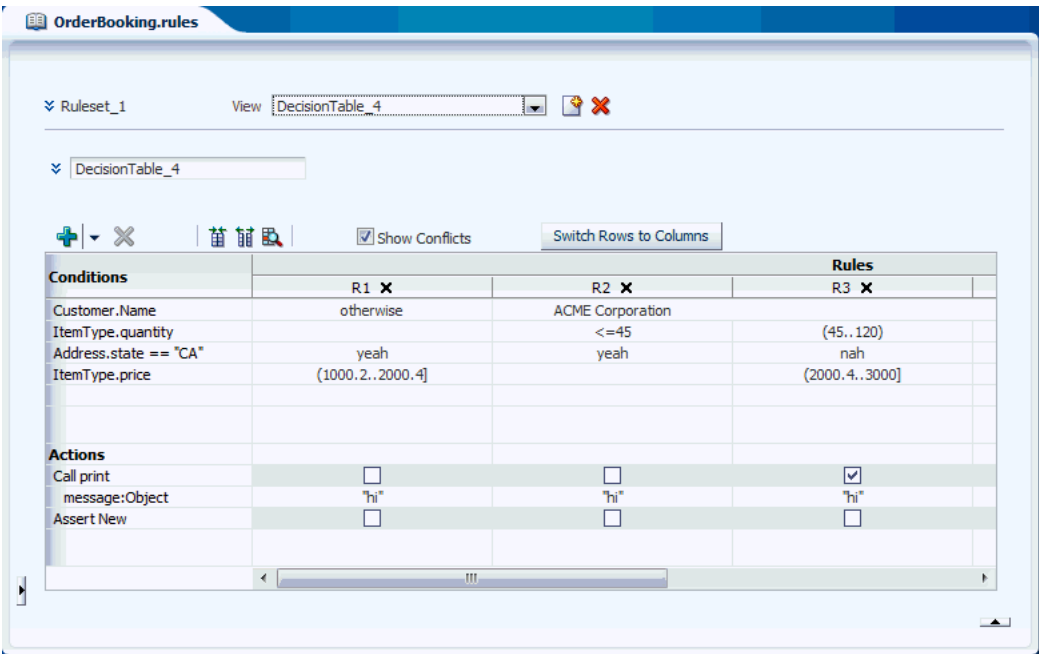
- Actions in a rule, as shown in Figure 25–3.

Figure 25–3 Actions in a Rule

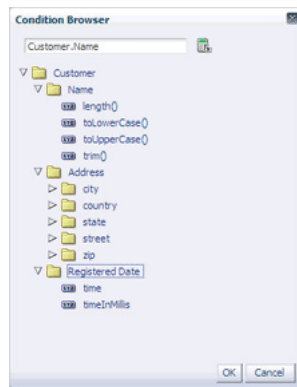


- Decision tables, as shown in Figure 25–4.

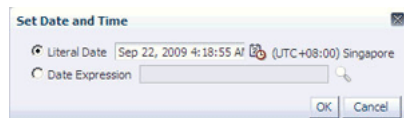
Figure 25–4 Decision Tables



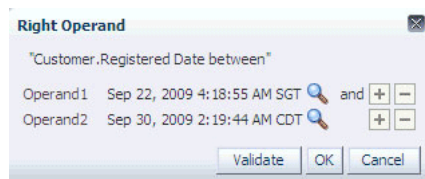
- Sets effective dates and priorities for rulesets and rules.
- Provides support for user-defined operators.
- Provides a Condition Browser pop-up to display the left or right value options, as shown in Figure 25–5.

Figure 25–5 Condition Browser

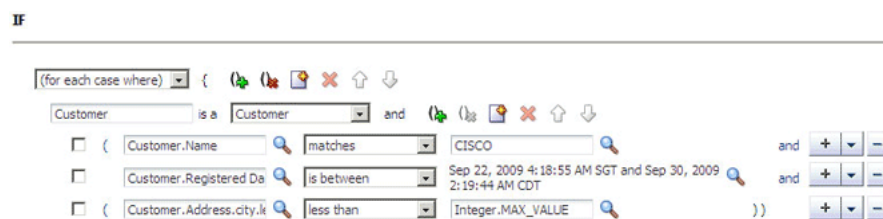
- Provides a Date Browser for selecting date types, as shown in [Figure 25–6](#).

Figure 25–6 Date Browser

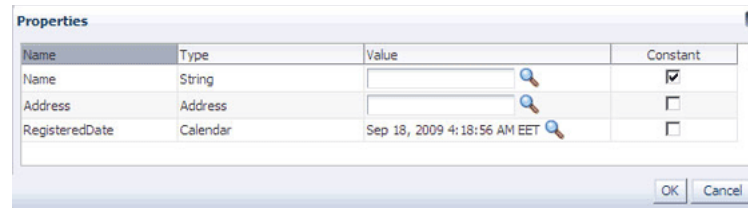
- Provides a Right Operand browser to handle multiple right-hand side expressions, as shown in [Figure 25–7](#).

Figure 25–7 Right Operand Browser

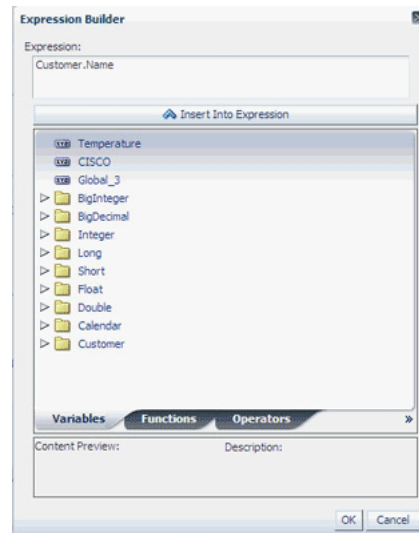
- Provides support for nested rules, as shown in [Figure 25–8](#).

Figure 25–8 Nested Rules Support

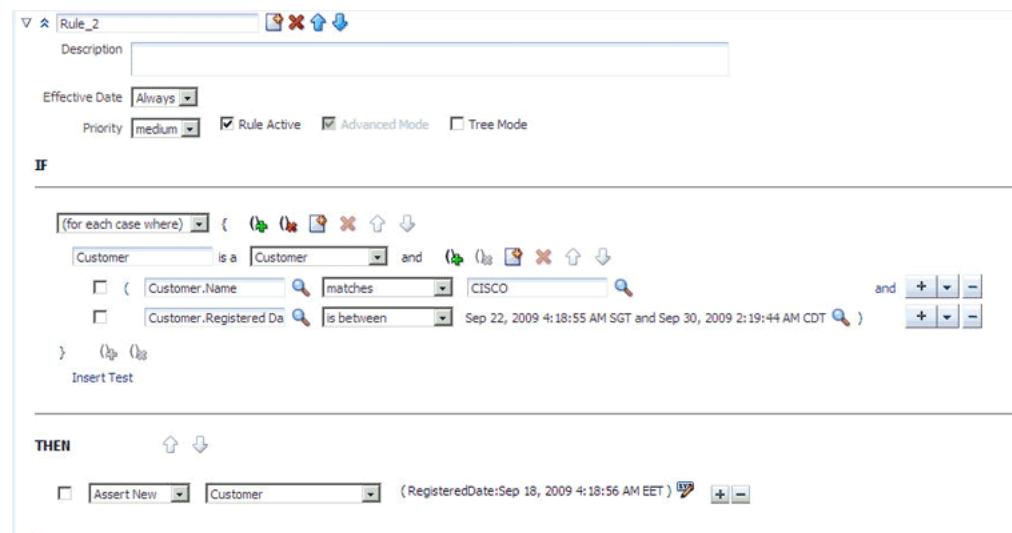
- Provides the Properties browser for editing properties of a rule action, as shown in [Figure 25–9](#).

Figure 25–9 Properties Browser

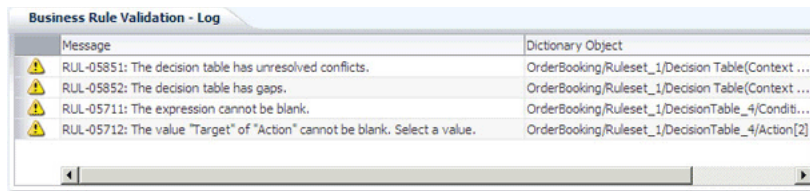
- Provides an Expression Builder window to build custom expressions, as shown in [Figure 25–10](#).

Figure 25–10 Expression Builder Window

- Provides **Advanced Mode** features for working with patterns and advanced actions, as shown in [Figure 25–11](#).

Figure 25–11 Advanced Mode Features

- Provides a Validation panel to manage error messages, as shown in [Figure 25–12](#).

Figure 25–12 Validation Panel to Manage Error Messages

Note: Once all the edits are done, the component user is responsible for saving the ruleset.

25.2.2 How to Create and Run a Sample Application by Using the Rules Editor Component

This section lists the steps for creating and running a sample application by using the Rules Editor component.

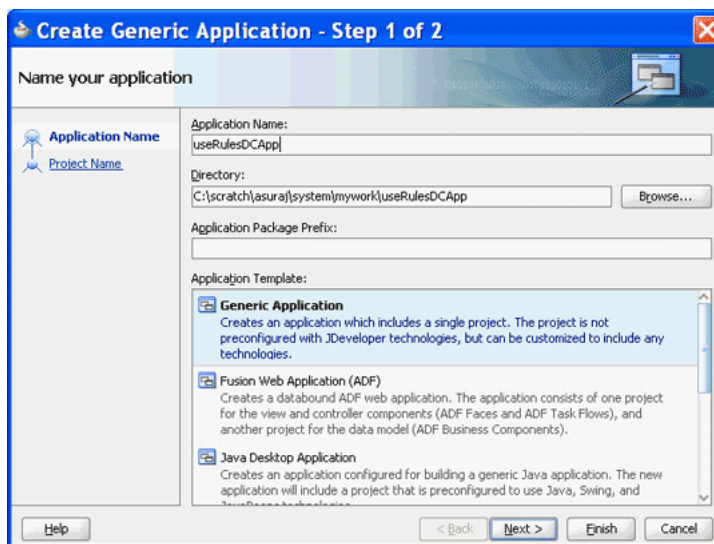
The prerequisite for using the Rules Editor component to create ADF-based Web applications is having a running installation of Oracle SOA Suite and Oracle JDeveloper on your computer.

To create a sample application by using the Rules Editor:

The first task is to create a sample application.

The steps are:

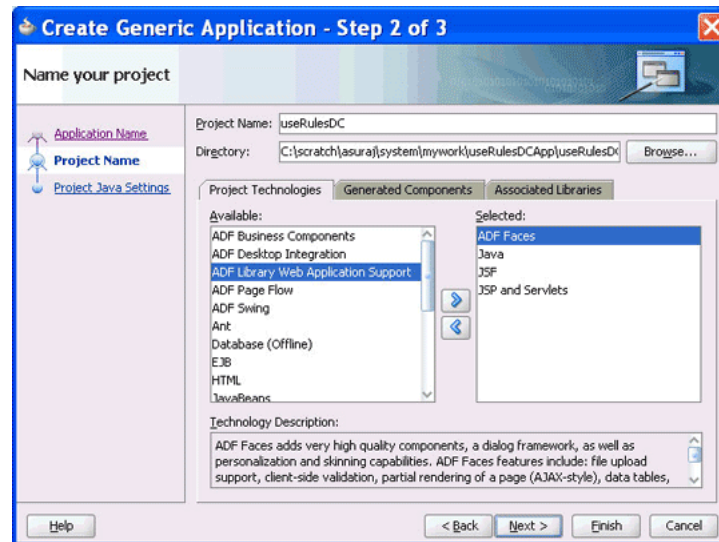
1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** and then **Generic Application** to create an application.
3. Enter a name for the application in the **Application Name** field, for example, `useRulesDCApp`, and click **Next** as shown in [Figure 25–13](#).

Figure 25–13 Creating a Generic Application

4. Enter `useRulesDC` in the **Project Name** field and ensure that **ADF Faces** is selected in the **Project Technologies** tab as shown in [Figure 25–14](#).

Click **Finish** to create the project.

Figure 25–14 Creating a Project

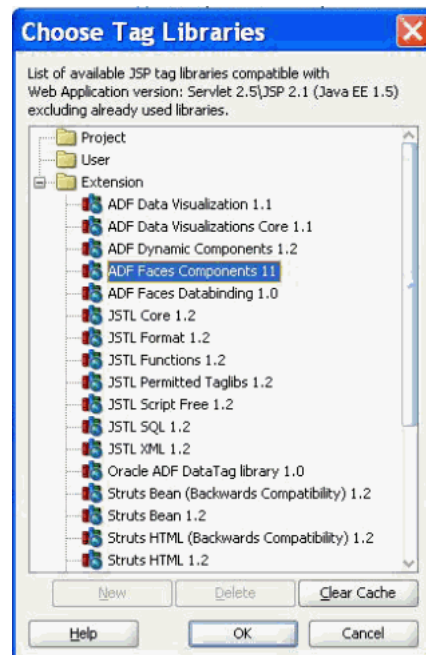


5. Right-click the **useRulesDC** project in the Application Navigator of Oracle JDeveloper, and select **Project Properties** to display the **Project Properties** dialog box.

In the Project Properties dialog box:

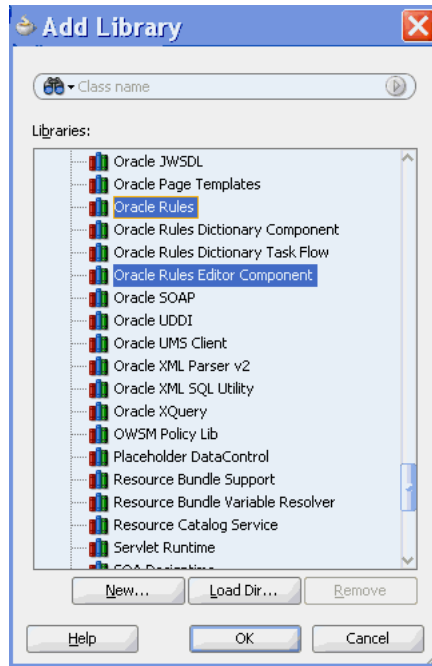
- a. Click **JSP Tag Libraries** from the left panel.
- b. Click **Add** and select **ADF Faces Components** from the Extension list in the Choose Tag Libraries dialog box, and click **OK** as shown in [Figure 25–15](#).

Figure 25–15 Choosing Tag Libraries



- c. Click **Libraries and Classpath** from the left panel and click the **Add Library** button to display the Add Library dialog box.
- d. Click **Oracle Rules** and **Oracle Rules Editor Component** from the Extension list and then click **OK** as shown in [Figure 25–16](#).

Figure 25–16 *Selecting Oracle Rules and Rules Editor Component*

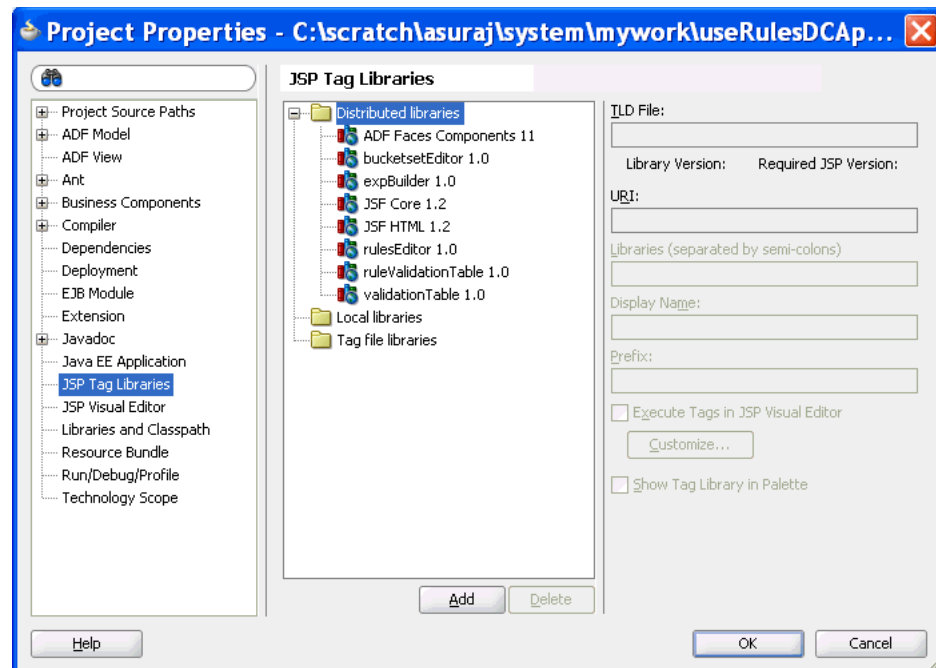


This adds the Rules SDK and the Rules Editor Component tag libraries to the project.

- e. Click **OK** to close the Project Properties dialog box.
6. Select **Save All** from the Oracle JDeveloper **File** menu to save the project.

You have to ensure that all the required tag libraries are added:

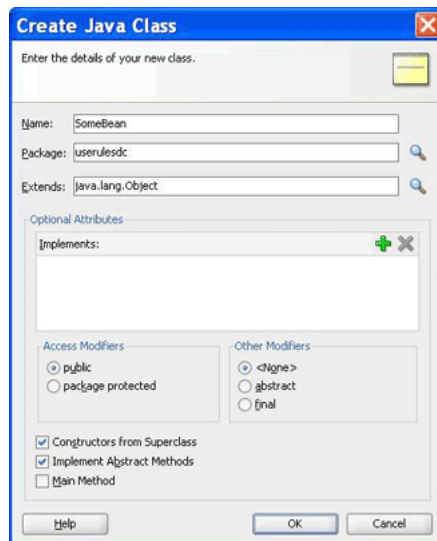
1. Right-click the **useRulesDC** project in the Application Navigator of Oracle JDeveloper and select **Project Properties** to display the Project Properties dialog box.
2. Click **JSP Tag Libraries** from the left panel and check if all the tag libraries are added as shown in [Figure 25–17](#).

Figure 25–17 Checking the Required Tag Libraries**To create the RuleSetModel object:**

The Rules Editor component requires a `oracle.bpel.rulesdc.model.impl.RuleSetModel` object. The component uses this object to read the rules and the decision tables that exist in the ruleset. So, the next task is to create a managed bean called `SomeBean.java` that creates a `RuleSetModel` object.

The steps are:

1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** to display the **New Gallery** dialog box.
3. In the **New Gallery** dialog box, select **Java** under General from the Categories panel. Ensure that **Java Class** under Items is selected and click **OK** to display the **Create Java Class** dialog box.
4. Enter the name of the Java class, for example `SomeBean.java`, and click **OK** to create the Java class in your project as shown in [Figure 25–18](#).

Figure 25–18 Creating a Java Class

5. In `SomeBean.java`, provide a method that returns the `RuleSetModel` object. You must specify the location of the rules file here. The following is a sample of the `SomeBean.java` file:

```
package userulesdc;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Reader;

import java.io.Writer;

import java.util.ArrayList;
import java.util.List;

import oracle.bpel.rulesdc.model.impl.RuleSetModel;
import oracle.rules.sdk2.dictionary.RuleDictionary;
import oracle.rules.sdk2.exception.SDKException;
import oracle.rules.sdk2.exception.SDKWarning;
import oracle.rules.sdk2.ruleset.RuleSet;
import oracle.rules.sdk2.ruleset.RuleSetTable;

public class SomeBean {
    private static final String RULES_FILE = "<your rules file here>";
    private RuleSetModel ruleSetModel = null;

    public RuleSetModel getRuleSetModel() {
        if (ruleSetModel != null)
            return ruleSetModel;
        //cache ruleSetModel instead of re-creating it each time

        Reader reader = null;
        try {
            reader =
                new FileReader(new File(RULES_FILE));
        } catch (FileNotFoundException e) {
```

```

        //LOG.severe(e);
        System.err.println(e);
    }

    RuleDictionary dict = null;

    try {
        dict = RuleDictionary.readDictionary(reader, null);
    } catch (SDKException e) {
        System.err.println(e);
    } catch (FileNotFoundException e) {
        System.err.println(e);
    } catch (IOException e) {
        System.err.println(e);
    }
    if (reader != null) {
        try {
            reader.close();
        } catch (IOException ioe) {
        }
    }
}

//get the ruleSetTable from the RuleDictionary object
RuleSetTable ruleSetTable = dict.getRuleSetTable();

//get the first ruleSet from the ruleSetTable
RuleSet ruleSet = ruleSetTable.get(0);
//create a RuleSetModel object and pass this to the rulesDC

ruleSetModel = new RuleSetModel(ruleSet) ;
return ruleSetModel;
}

//refer to Rules SDK documentation for saving a dictionary also
//because this code does not take care of saving linked dictionaries
public static boolean saveDictionary(RuleDictionary dict,
                                    String ruleFileName) {
    Writer writer = null;
    try {
        writer = new FileWriter(new File(ruleFileName));
        dict.writeDictionary(writer);

    } catch (SDKException e) {
        System.err.println(e);
        return false;
    } catch (FileNotFoundException e) {
        System.err.println(e);
        return false;
    } catch (IOException e) {
        System.err.println(e);
        return false;
    } finally {
        if (writer != null) {
            try {
                writer.close();
            } catch (IOException ioe) {
                return false;
            }
        }
    }
}

```

```

        return true;
    }

    public static void updateDictionary(RuleDictionary dict) {
        if (dict == null)
            return;

        List<SDKWarning> warnings = new ArrayList<SDKWarning>();
        try {
            dict.update(warnings);
            if (!warnings.isEmpty()) {
                for (int i = 0; i < warnings.size(); i++)
                    System.out.println("warnings: " +
                                      warnings.get(i).getLocalizedMessage());
            }
        } catch (SDKException sdkEx) {
            sdkEx.printStackTrace();
        }
    }

    //You can call this method from your "Save" button
    public void saveDictionary() {

        RuleDictionary dict = this.getRuleSetModel().getRuleSet().getDictionary();
        if (dict != null) {
            //update the dictionary before saving it
            updateDictionary(dict);
            saveDictionary(dict, RULES_FILE);
        }
    }

    //call the validation method on the ruleSetModel to update the Validation Panel

    public void validate() {
        if (this.ruleSetModel == null)
            return;

        this.ruleSetModel.validate();
    }

```

- Open the faces-config.xml file in Overview mode and click the + button under Managed Beans to display the **Create Managed Bean** dialog box. Point to SomeBean.java by providing the Bean Name as someBean and the Scope as session as shown in [Figure 25-19](#).

Figure 25-19 Specifying the Bean Name and Scope



The ADF/JSF framework makes multiple calls to SomeBean.java to render the user interface. For example, someBean.ruleSetModel is called multiple times.

So, it is better to create the `RuleSetModel` object once, cache it, and return it each time instead of re-creating it.

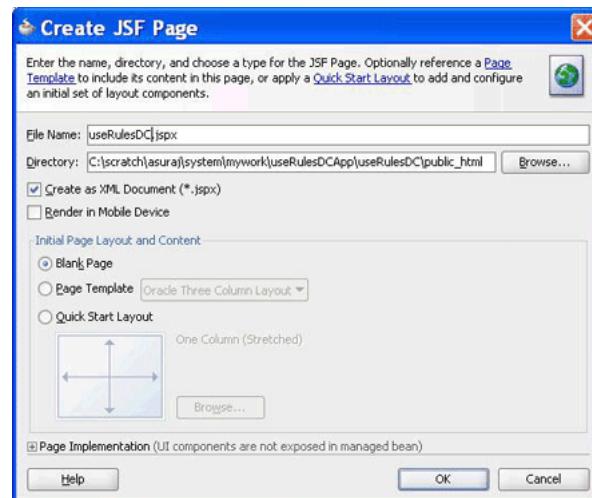
To create the .jspx file for the Rules Editor Component tag:

The next task is to create the .jspx file to include the Rules Editor component tag.

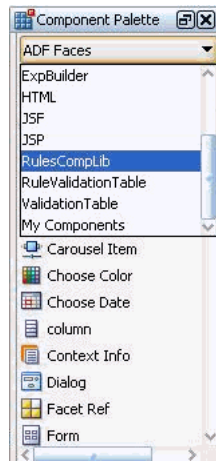
The steps are:

1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** to display the **New Gallery** dialog box.
3. In the **New Gallery** dialog box, select **JSF** under Web Tier from the Categories panel.
4. Select **JSF Page** under Items and click **OK** to display the **Create JSF Page** dialog box.
5. In the **Create JSF Page** dialog box, enter `useRulesDC.jsxpx` as the file name as shown in Figure 25–20.

Figure 25–20 Creating the JSF Page File



`RulesCompLib` in the component palette of Oracle JDeveloper is displayed as shown in Figure 25–21.

Figure 25–21 Rules Editor Component Library in the Component Palette

This is because you have added the Rules Editor Component tag library when creating the sample application.

6. Select **RulesCompLib** to view the Rulesdc tag. You can drag and drop the Rulesdc tag into the .jspx file. You can also add the Rulesdc tag in the .jspx file manually as shown:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
  xmlns:rdc="http://xmlns.oracle.com/bpel/rules/editor">
  <jsp:directive.page contentType="text/html; charset=UTF-8"/>
  <f:view>
    <af:document title="Sample Rules Editor App" id="d1">
      <af:form id="f1">
        <af:panelStretchLayout id="psl1" inlineStyle="margin:15px;"
          partialTriggers="cb1 cb3">
          <f:facet name="center">
            <rdc:rulesdc rulesetModel="#{someBean.ruleSetModel}"
              viewOnly="false" discloseRules="true"
              genericAction="true" genericPattern="true"
              dtColumnPageSize="6" id="r1" dateStyle="yyyy-MM-dd"
              timeStyle="HH-mm-ss"></rdc:rulesdc>
          </f:facet>
          <f:facet name="top">
            <af:panelGroupLayout id="pgl2" layout="horizontal">
              <af:commandButton text="Save Dictionary"
                action="#{someBean.saveDictionary}" id="cb1"/>
              <af:spacer width="10" height="10" id="s5"/>
              <af:commandButton text="Validate" id="cb3"
                action="#{someBean.validate}"
                partialSubmit="true"/>
            </af:panelGroupLayout>
          </f:facet>
        </af:panelStretchLayout>
      </af:form>
    </af:document>
  </f:view>
</jsp:root>
```

To refer to the oracle.rules and the oracle.soa.rules_editor_dc.webapp shared libraries:

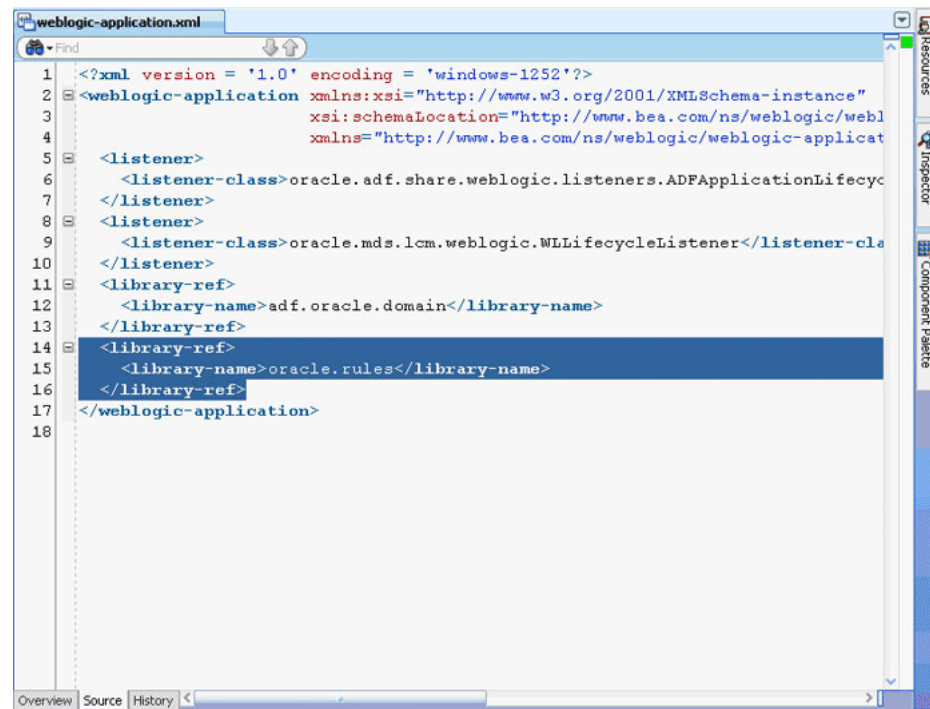
After creating the .jspx file, you must refer to the oracle.rules and oracle.soa.rules_editor_dc.webapp shared libraries from the weblogic-application.xml file.

The steps are:

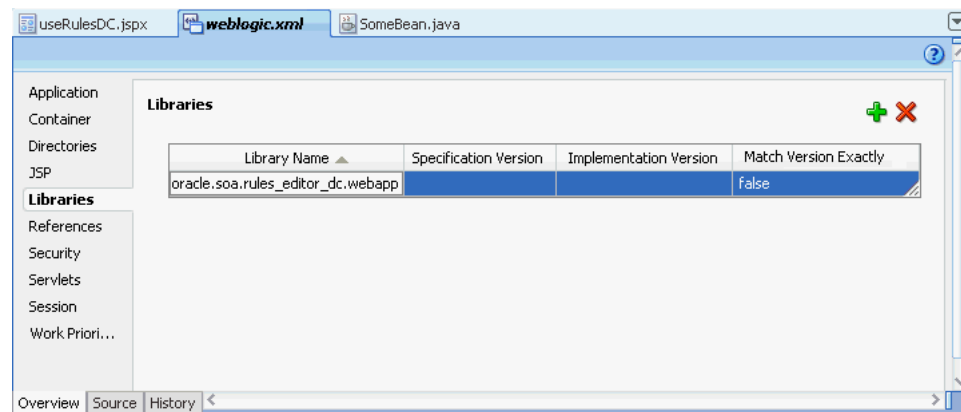
1. In Oracle JDeveloper, open the weblogic-application.xml file by browsing to Application Resources, then Descriptors, and then META-INF.
2. Add the following lines to refer to the oracle.rules shared library as shown in Figure 25-22.

```
<library-ref>
<library-name>oracle.rules</library-name>
</library-ref>
```

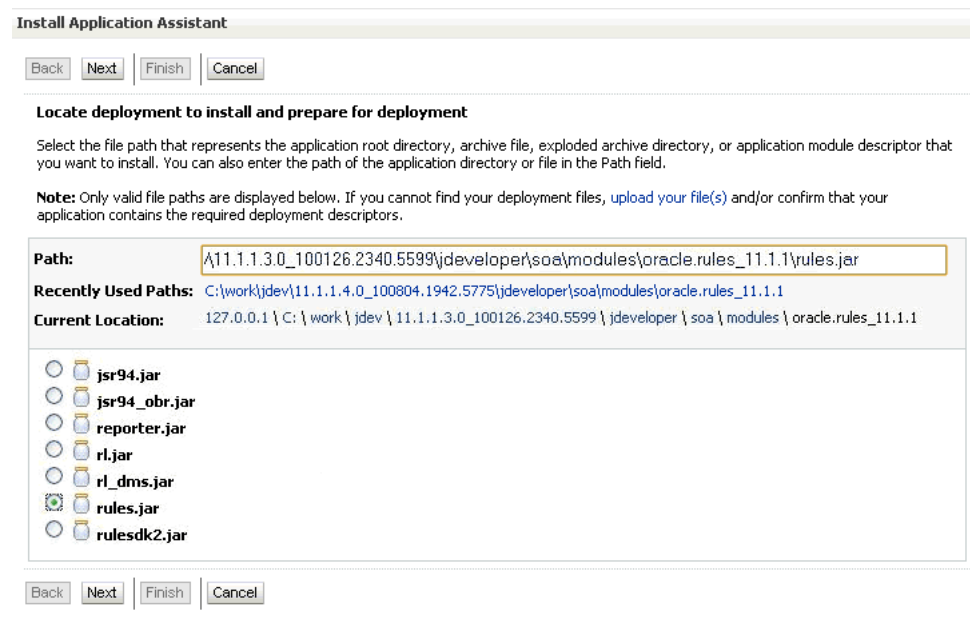
Figure 25-22 Referring to the oracle.rules Shared Library



3. In Oracle JDeveloper,
 - a. From the **File** menu, select **New** to display the New Gallery dialog box.
 - b. In the New Gallery dialog box, select **Deployment Descriptors** under **General** from the Categories panel.
 - c. Select **Weblogic Deployment Descriptor** under **Items** and click **OK** to display the Create Weblogic Deployment Descriptor dialog box.
 - d. Select **weblogic.xml** from the list and click **Finish**.
 - e. In Oracle JDeveloper, in the Overview mode of weblogic.xml, select **Libraries** from the left panel and enter **oracle.soa.rules_editor_dc.webapp** as the library name as shown in Figure 25-23.

Figure 25–23 Adding the Rules Editor Component Library

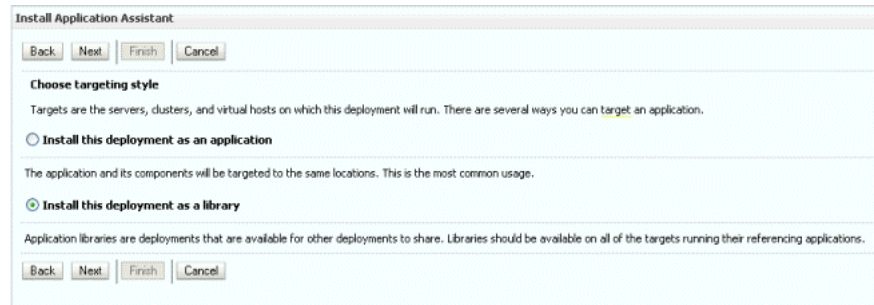
- f. Click **Save All**.
4. Deploy the `oracle.rules` shared library to the embedded Weblogic server:
 - a. Launch WLS console (<http://host:port/console/login/LoginForm.jsp>). Ensure that the Weblogic embedded server on Oracle JDeveloper is running.
 - b. Select **Deployments** and click **Install** to display the Install Application Assistant page.
 - c. Select `<JDEV_INSTALL>/jdeveloper/soa/modules/oracle.rules_11.1.1/rules.jar` and click **Finish** as shown in [Figure 25–24](#).

Figure 25–24 Deploying the oracle.rules Shared Library

5. Deploy the `oracle.soa.rules_editor_dc.webapp` shared library to the Weblogic server:
 - a. In the Weblogic console, select **Deployments** and click **Install** to display the Install Application Assistant page.

- b. Select `<JDEV_INSTALL>/jdeveloper/soa/modules/oracle.soa.rules_editor_dc.webapp_11.1.1/oracle.soa.rules_editor_dc.webapp.war` and click **Next**.
- c. Select **Install this deployment as a library** and click **Finish** as shown in Figure 25-25.

Figure 25-25 Deploying oracle.soa.rules_editor_dc.webapp Shared Library



`oracle.soa.rules_editor_dc.webapp` is added to the list of deployments as shown in Figure 25-26.

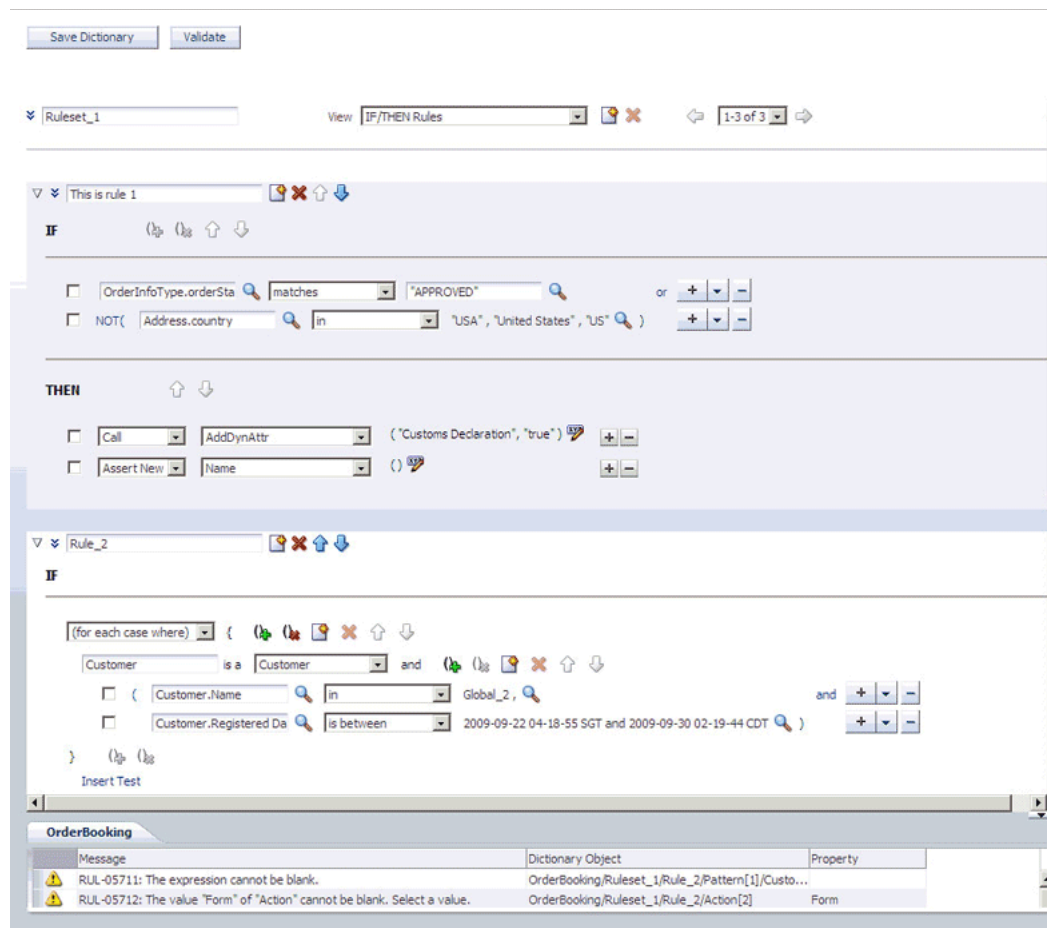
Figure 25-26 oracle.soa.rules_editor_dc.webapp Added to the Deployment List

☐	oracle.bi.adf.webcenter.slib(1.0,11.1.1.2.0)	Active		Library	100
☐	oracle.dconfig-infra(11,11.1.1.1.0)	Active		Library	100
☐	oracle.yf.system.filter	Active		Library	100
☐	oracle.jsp.next(11.1.1,11.1.1)	Active		Library	100
☐	oracle.pwdgen(11.1.1,11.1.1.2.0)	Active		Library	100
☐	oracle.soa.rules_editor_dc.webapp(11.1.1,11.1.1)	Active		Library	100
☐	oracle.wsm.seedpolicies(11.1.1,11.1.1)	Active		Library	100
☐	oracle18n-adf(11,11.1.1.1.0)	Active		Library	100
☐	UDX(11,11.1.1.1.0)	Active		Library	100
☐	wsdl-wls	Active	✓ OK	Enterprise Application	5
☐	wsm-pm	Active	✓ OK	Enterprise Application	5

To run the sample Rules Editor application:

The last task is running the sample application.

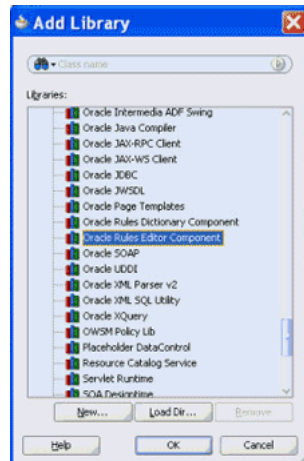
To run the sample application, from Oracle JDeveloper, right-click the `useRulesDC.jspx` file, and select **Run**. This starts the sample application on a Web browser as shown in Figure 25-27.

Figure 25–27 Running the Sample Application

25.2.3 How to Deploy a Rules Editor Application to a Standalone Weblogic Server

When you are ready to deploy your application EAR file to the standalone Weblogic server, perform the following:

1. Launch the Weblogic server console (<http://host:port/console/login/LoginForm.jsp>) and ensure that `oracle.rules` is displayed in the deployments list.
2. Ensure that `oracle.soa.rules_editor_dc.webapp` is displayed in the deployments list. If this is not displayed, click **Install** and select the `<JDEV_INSTALL>/jdeveloper/soa/modules/oracle.soa.rules_editor_dc.webapp_11.1.1/oracle.soa.rules_editor_dc.webapp.war` file.
3. Open Oracle JDeveloper.
4. Right-click the project name in the Application Navigator and select **Project Properties**.
5. Select **Libraries and Classpath** from the left panel and click **Add Library**.
6. In the **Add Library** dialog box, select **Oracle Rules Editor Component** and click **OK** as shown in Figure 25–28.

Figure 25-28 Adding the Oracle Rules Editor Component

This step enables you to refer to these libraries, but does not deploy these libraries by default. Therefore, the jars are not included in your project WAR file.

7. In the project that has to be deployed (where you create the EAR file):

a. Add the following lines to the weblogic-application.xml:

```
<library-ref>
  <library-name>oracle.rules</library-name>
</library-ref>
```

b. Add the following lines to weblogic.xml in the project WAR file:

```
<library-ref>
  <library-name>oracle.soa.rules_editor_dc.webapp</library-name>
</library-ref>
```

c. Deploy the EAR file in the Weblogic server.

For more information about creating an EAR file, see "How to Create an EAR File for Deployment" in *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.

25.2.4 What You May Need to Know About the Custom Permissions for the Rules Editor Component

For the role based authorization, Rules DC implements custom JAAS permissions (extending `oracle.adf.share.security.authorization.ADFPermission` class to ensure that the permission can be used by ADF Security).

If a Rules Editor application supports ADF security, which means there is support for role-based authentication and authorization, then security is enforced by implementing custom JAAS permissions (by extending the `oracle.adf.share.security.authorization.ADFPermission` class to ensure that the permission can be used by ADF Security). You have to create ADF security policies by granting the following permissions to the user roles based on your application requirement:

- `oracle.rules.adf.permission.AddRulePermission`: Displays **Add Rule** button; if the permission is not granted, the **Add Rule** button is not visible to the user.

- `oracle.rules.adf.permission.DeleteRulePermission`: Displays the **Delete Rule** button; if the permission is not granted, the **Delete Rule** button is not visible to the user.
- `oracle.rules.adf.permission.EditRulePermission`: Displays the **Edit Rule** button for rules inside a ruleset; if the permission is not granted, then the rules are "View-Only".
- `oracle.rules.adf.permission.AddDTPPermission`: Displays the **Add Decision Table** button; if the permission is not granted, the **Add Decision Table** button is not visible to the user.
- `oracle.rules.adf.permission.DeleteDTPPermission`: Displays the **Delete Decision Table** button; if the permission is not granted, the **Delete Decision Table** button is not visible to the user.
- `oracle.rules.adf.permission.EditDTPPermission`: Displays the **Edit Decision Table** button for decision tables within a ruleset; if the permission is not granted, the decision tables are "View-Only".
- `oracle.rules.adf.permission.RulesEditorPermission`: A global permission that sets all the preceding permissions to "true".

For example, to grant the delete rule permission to a role, specify the following code in the `jazn-data.xml` file of the application:

```
<grant>
  <grantee>
    <principals>
      <principal>
        <class>oracle.security.jps.service.policystore.ApplicationRole</class>
        <name>role2</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.rules.adf.permission.DeleteRulePermission</class>
      <name>DeleteRulePermission</name>
      <actions>access</actions>
    </permission>
  </permissions>
</grant>
```

If you do not want to use the individual permissions, such as `AddRulePermission` or `DeleteRulePermission`, you can just set the `RulesEditorPermission` in the `jazn-data.xml` file to set global permissions.

25.2.5 What You May Need to Know About the Supported Tags of the Rules Editor Component

This section lists the tags and attributes that are supported by the Rules Editor component.

[Table 25–1](#) lists the supported facets.

Table 25–1 Supported Facets of the Rules Editor Component

Name	Description
patternDisplay	Used to render specific user interfaces. This facet is used to display the rule condition and pattern (in advanced mode), which is the "IF" portion of the rule.
actionDisplay	Used to render specific user interfaces. This facet is used to display the rule action, which is the "THEN" portion of the rule.

Table 25–2 lists the supported attributes.

Table 25–2 Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
dateStyle	java.lang.String	no	Gets from the locale	yes	If specified, the date style is used in all inputDate components, for example, yyyy.MM.dd.
decimalSeparator	java.lang.Character	no	Based on Locale	yes	Specifies the decimal separators. This is used in Number Formatting. If specified, this attribute overrides the decimal separator based on locale.
disableRuleSetName	java.lang.Boolean	no	false	yes	If true, the editable ruleset name is disabled. This attribute is used only when displayRuleSetName is set to true.
discloseRules	java.lang.Boolean	no	false	yes	If true, all the rules in the ruleset are expanded. If "false", all the rules are collapsed.
displayRuleSetEffDate	java.lang.Boolean	no	true	yes	If true, the Rules Editor component renders the user interface for displaying the effective dates for the RuleSet.
displayRuleSetName	java.lang.Boolean	no	true	yes	Displays the editable ruleset name by default. You can choose to hide this by setting it to false.

Table 25–2 (Cont.) Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
dtColumnPageSize	java.lang.Integer	no	5	yes	Specifies the number of columns to be displayed at a time in a decision table. This works only when rules are columnar.
dtHeight	java.lang.Integer	no	16	yes	Number of rows to be displayed at a time in the decision table. A scroll bar is displayed if the number of rows increases over the specified height.
genericAction	java.lang.Boolean	no	true	yes	If true, the Rules Editor component renders the user interface for displaying the THEN part, which is Actions. If "false", then the "actionDisplay" facet must be passed to the Rules Editor component. The facet must contain the user-defined user interface. The facet has access to the ActionModel.
genericPattern	java.lang.Boolean	no	true	yes	If true, the Rules Editor component renders the user interface for displaying the IF part, which is Conditions and Patterns (in Advanced Mode). If false, then the "patternDisplay" facet must be passed to the Rules Editor component. The facet must contain the user-defined user interface. The facet has access to the RuleModel and SimpleTestModel.
groupingSeparator	java.lang.Character	no	Based on Locale	yes	Specifies the grouping separators. This is used in Number Formatting. If specified, this attribute overrides the grouping separator based on locale.

Table 25–2 (Cont.) Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
locale	java.util. Locale	no	Locale.getDefault()	yes	Used for Localization
ruleModel	java.lang. String	no	oracle. bpel.rules dc.model. impl. RuleModel	yes	Used to customize the default RuleModel. You can extend the RuleModel class to override certain methods.
rulesetModel	oracle.bpel. rulesdc.model. interfaces. RuleSetInterface	yes	-	Only EL	Wrapper around the Rules SDK ruleset object. You can use the RuleSetModel object supplied as part of the Rules Editor Component jar file (adflibRulesDC.jar).
rulesPageSize	java.lang. Integer	no	5	yes	Specifies the number of rules to be displayed in a page. It is used in IF/THEN rules pagination.
showDTButtons	java.lang. Boolean	no	true	yes	Displays the Add and Delete Decision Table links by default. You can choose to hide this by setting this to false.
showValidationPanel	java.lang. Boolean	no	true	yes	Displays the validation panel by default. You can choose to hide this by setting this to false.
simpleTestModel	java.lang. String	no	oracle. bpel.rules dc.model. impl. SimpleTest Model	yes	Used to customize the default SimpleTestModel. You can extend the SimpleTestModel class to override certain methods.
timeStyle	java.lang. String	no	Gets from the locale	yes	If specified, the time style is used in all inputDate components, for example HH:mm:ss.
timezone	java.util. TimeZone	no	TimeZone. getDefault ()	yes	Used for Localization

Table 25–2 (Cont.) Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
viewOnly	java.lang. Boolean	no	true	yes	If "true", in the "viewOnly" mode, you can view the existing rules in the ruleset. If "false", which is the "edit" mode, you can add new rules and edit existing rules.

25.3 Using the Oracle Business Rules Dictionary Editor Declarative Component

This section discusses the Oracle Business Rules Dictionary Editor declarative component. It also provides information on how to create and run an application using the Rules Dictionary Editor component, and then deploy the application. In addition, this section lists the supported tags and the localization process for the application.

25.3.1 Introduction to the Oracle Business Rules Dictionary Component

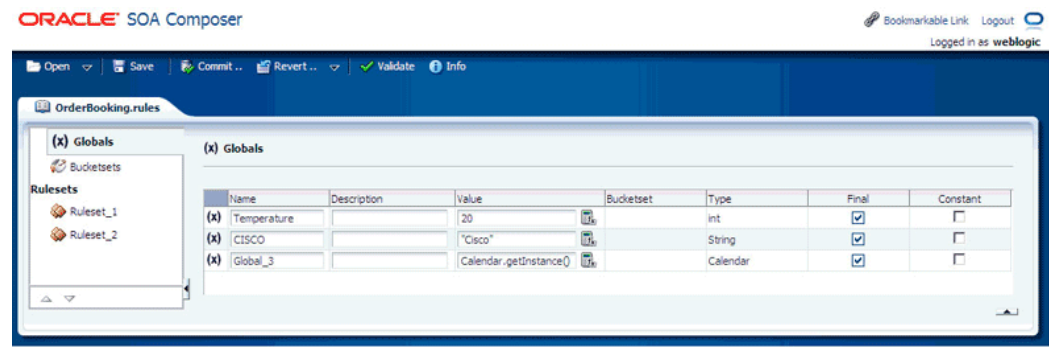
The Oracle Business Rules Dictionary Editor is a composite declarative component that can be embedded in any ADF-based Web application. It enables you to edit business rules metadata artifacts, such as Globals, Bucketsets, and Rulesets, by using the Rules SDK2 API.

Note: You should not confuse Rules Dictionary Editor with Rules Editor. Rules Editor is used to edit rules inside a specified ruleset. In fact, Rules Editor is embedded within Rules Dictionary Editor. For more information about Rules Editor, see [Section 25.2, "Using the Oracle Business Rules Editor Declarative Component."](#)

The Rules Dictionary Editor Task Flow uses the Rules Dictionary Editor Component to create applications. Typically, you should either use the Rules Dictionary Editor component or the Rules Dictionary Editor task flow, but not both. For more information on Rules Dictionary Editor Task Flow, see [Section 25.4, "Using the Oracle Business Rules Dictionary Task Flow."](#)

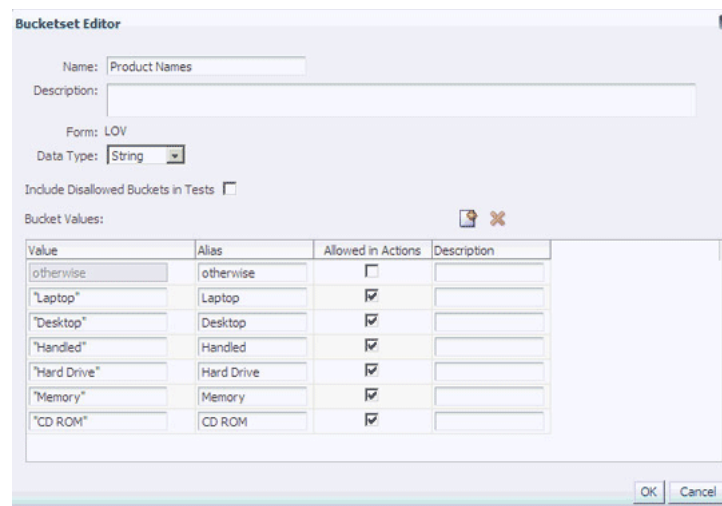
The Rules Dictionary Editor component performs the following:

- Edits Globals or Variables that have the `final` attribute set to `true` by using the Globals Editor, as shown in [Figure 25–29](#).

Figure 25–29 Globals Editor

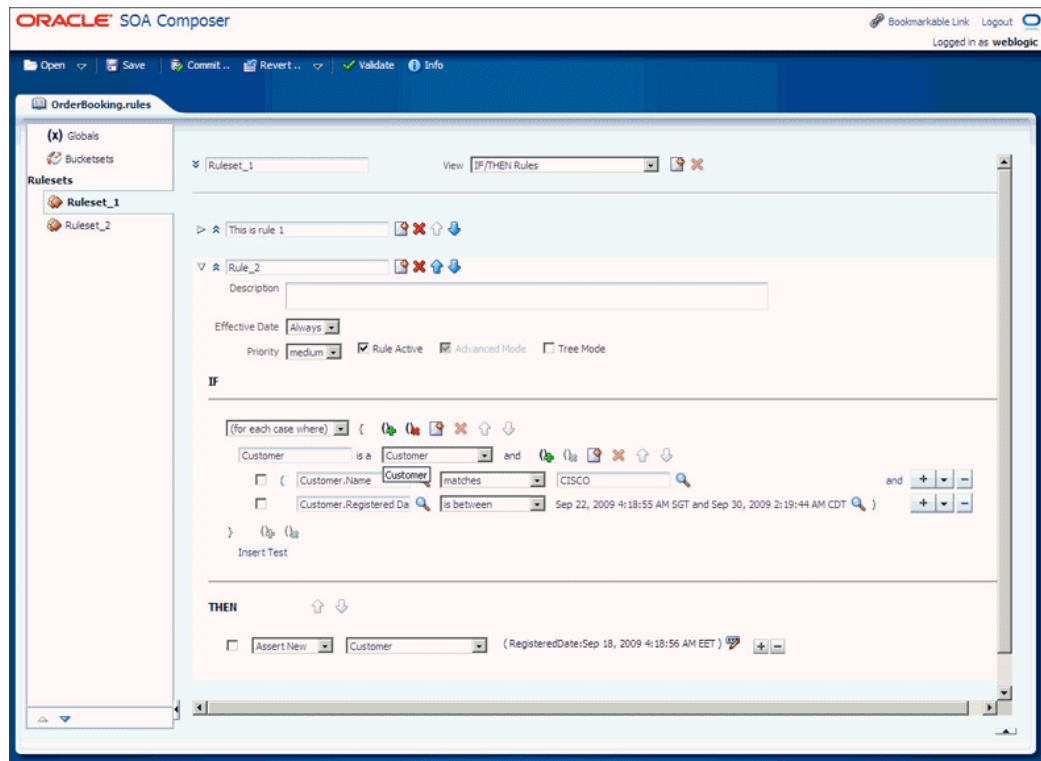
Globals Editor enables you to edit only the Name, Description, and Value of Globals. It does not allow creation or deletion of Globals. However, it supports validation of Globals.

- Edits Bucketsets by using the Bucketset Editor as shown in [Figure 25–30](#).

Figure 25–30 Bucketset Editor

Bucketset Editor enables you to perform CRUD (create, read, update, and delete) operations on Bucketsets and buckets inside a Bucketset. It also supports validation of Bucketsets.

- Edits Rulesets, as shown in [Figure 25–31](#).

Figure 25–31 Edits Rulesets

Rules Dictionary Editor enables you to edit only rules inside a selected ruleset. It does not allow creation or deletion of rulesets.

25.3.2 How to Create and Run a Sample Application by Using the Rules Dictionary Editor Component

This section lists the steps for creating and running a sample application by using the Rules Dictionary Editor component.

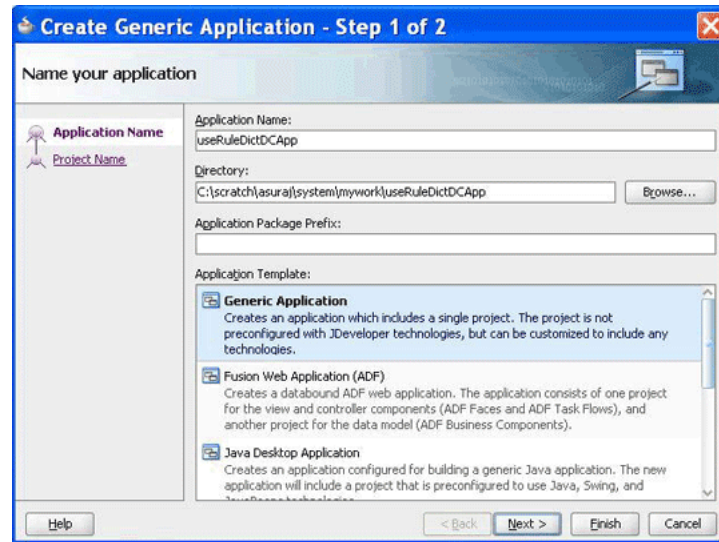
The prerequisite for using the Rules Dictionary Editor component to create ADF-based Web applications is having a running installation of SOA Suite and Oracle JDeveloper on your computer.

To create a sample application by using the Rules Dictionary Editor:

The first task is to create a sample application.

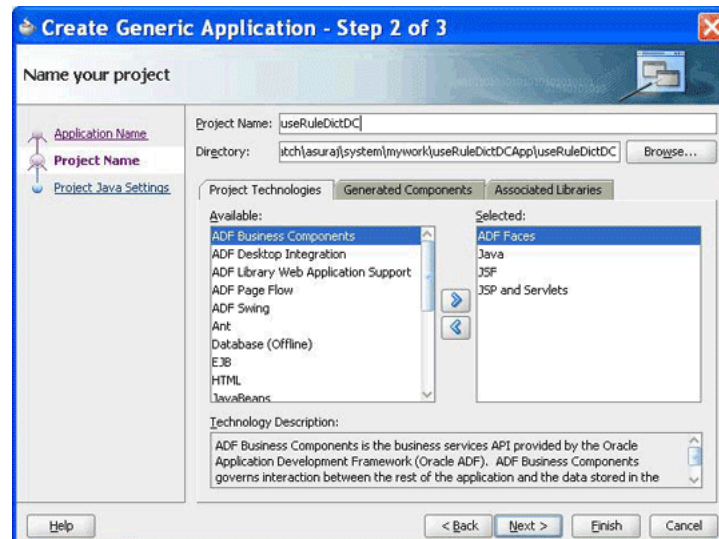
The steps are:

1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** and then **Generic Application** to create an application.
3. Enter a name for the application in the **Application Name** field, for example, useRuleDictDCApp, and click **Next** as shown in [Figure 25–32](#).

Figure 25–32 Creating a Generic Application

4. Enter `useRuleDictDC` in the **Project Name** field and ensure that **ADF Faces** is selected in the **Project Technologies** tab as shown in [Figure 25–33](#).

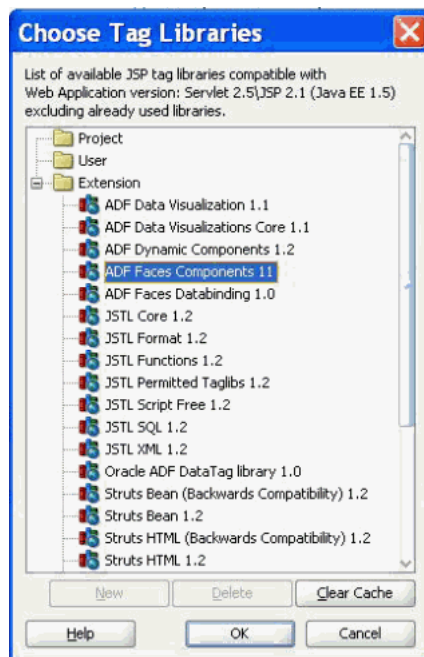
Click **Finish** to create the project.

Figure 25–33 Creating a Project

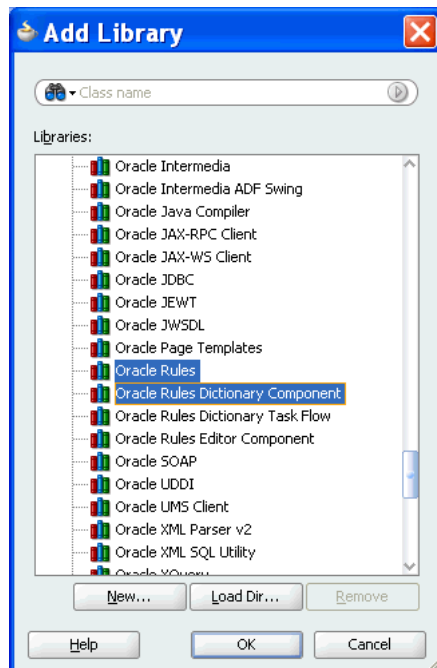
5. Right-click the `useRuleDictDC` project in the Application Navigator of Oracle JDeveloper, and select **Project Properties** to display the **Project Properties** dialog box.

In the Project Properties dialog box:

- a. Click **JSP Tag Libraries** from the left panel.
- b. Click **Add** and select **ADF Faces Components** from the extension list in the Choose Tag Libraries dialog box, and then click **OK** as shown in [Figure 25–34](#).

Figure 25–34 Choosing Tag Libraries

- c. Click **Libraries and Classpath** from the left panel and click the **Add Library** button to display the Add Library dialog box.
- d. Click **Oracle Rules** and **Oracle Rules Dictionary Component** from the Extension list and then click **OK** as shown in [Figure 25–35](#).

Figure 25–35 Selecting Oracle Rules and Rules Dictionary Component

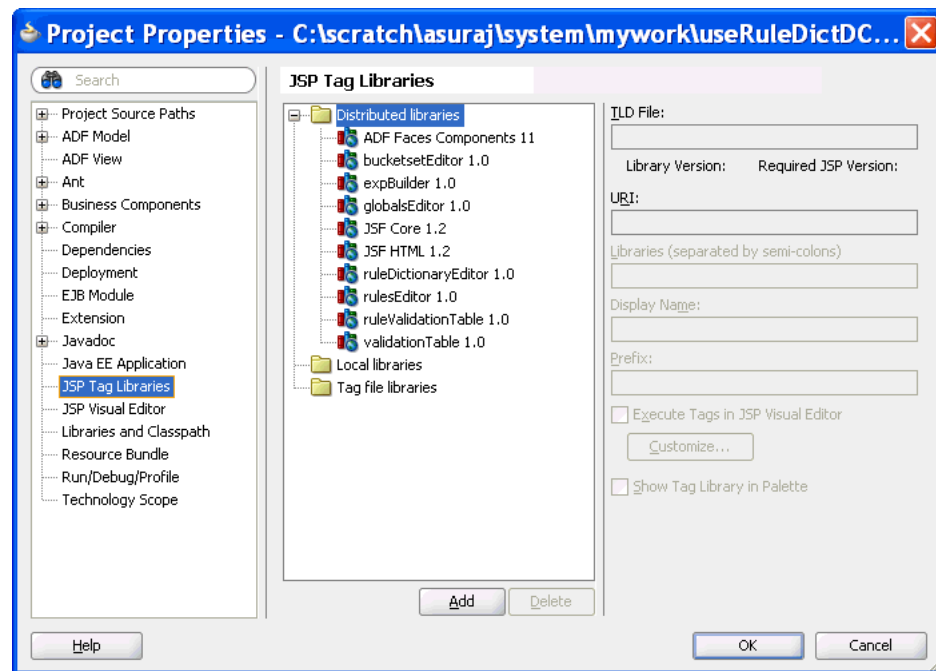
This adds the Rules SDK and the Rules Dictionary Editor tag libraries to the project.

- e. Click **OK** to close the Project Properties dialog box.
6. Select **Save All** from the Oracle JDeveloper **File** menu to save the project.

You have to ensure that all the required tag libraries are added:

1. Right-click the **useRuleDictDC** project in the Application Navigator of Oracle JDeveloper and select **Project Properties** to display the Project Properties dialog box.
2. Click **JSP Tag Libraries** from the left panel and check if all the tag libraries are added as shown in [Figure 25–36](#).

Figure 25–36 Checking the Required Tag Libraries for Rules Dictionary Editor

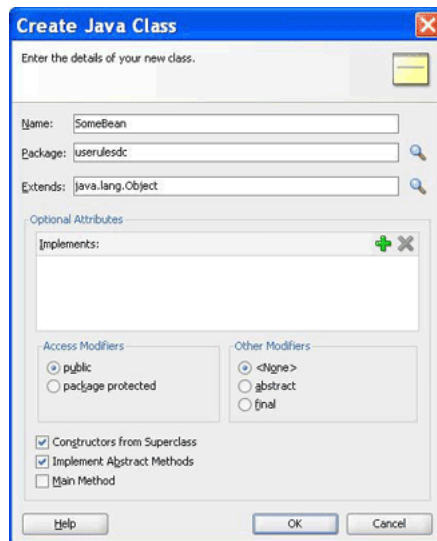


To create the RuleDictionaryModel object:

The Rules Dictionary Editor component requires a `oracle.bpel.ruledictionarydc.model.impl.RuleDictionaryModel` object. The component uses this object to read Globals, Bucketsets, and Rulesets information from the dictionary. So, the next task is to create a managed bean called `SomeBean.java` that creates a `RuleDictionaryModel` object.

The steps are:

1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** to display the **New Gallery** dialog box.
3. In the **New Gallery** dialog box, select **Java** under General from the Categories panel. Ensure that **Java Class** under Items is selected and click **OK** to display the **Create Java Class** dialog box.
4. Enter the name of the Java class, for example `SomeBean.java`, and click **OK** to create the Java class in your project as shown in [Figure 25–37](#).

Figure 25–37 Creating a Java Class

5. In `SomeBean.java`, provide a method that returns the `RuleDictionaryModel` object. You must specify the location of the rules file here. The following is a sample of the `SomeBean.java` file:

```
package useruledictdc;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Reader;

import java.io.Writer;

import java.util.ArrayList;
import java.util.List;

import oracle.bpel.ruledictionarydc.model.impl.RuleDictionaryModel;

import oracle.rules.sdk2.dictionary.DictionaryFinder;
import oracle.rules.sdk2.dictionary.RuleDictionary;
import oracle.rules.sdk2.exception.SDKException;
import oracle.rules.sdk2.exception.SDKWarning;

public class SomeBean {
    private RuleDictionaryModel ruleDictModel;
    private static final String RULES_FILE1 =
        "C:\\scratch\\asuraj\\system\\rules\\OrderBookinRules.rules";

    public SomeBean() {
        super();
    }

    public RuleDictionaryModel getRuleDictModel() {
        if (ruleDictModel != null)
            return ruleDictModel;
        //cache ruleDictModel instead of re-creating it each time
    }
}
```

```

        ruleDictModel = new RuleDictionaryModel(getRuleDictionary());
        return ruleDictModel;
    }

    public RuleDictionary getRuleDictionary() {

        Reader reader = null;
        try {
            reader = new FileReader(new File(RULES_FILE1));
        } catch (FileNotFoundException e) {
            //LOG.severe(e);
            System.err.println(e);
        }
        RuleDictionary dict = openRulesDict(reader, null);
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException ioe) {
            }
        }

        return dict;
    }

    private static RuleDictionary openRulesDict(Reader reader,
                                                DictionaryFinder finder) {

        RuleDictionary dict = null;

        try {
            dict = RuleDictionary.readDictionary(reader, finder);
        } catch (SDKException e) {
            System.err.println(e);
        } catch (FileNotFoundException e) {
            System.err.println(e);
        } catch (IOException e) {
            System.err.println(e);
        }
        catch (IllegalArgumentException e) {
            System.err.println(e);
        } finally {
        }

        return dict;
    }

    //refer to Rules SDK documentation for saving a dictionary also
    //because this code does not take care of saving linked dictionaries

    public static boolean saveDictionary(RuleDictionary dict,
                                        String ruleFileName) {

        if (dict == null || ruleFileName == null)
            return false;

        if (dict.isTransactionInProgress())
            System.out.println("Transaction in progress, cannot save
dictionary");

        Writer writer = null;

```

```
        try {
            writer = new FileWriter(new File(ruleFileName));
            dict.writeDictionary(writer);

        } catch (SDKException e) {
            System.err.println(e);
            return false;
        } catch (FileNotFoundException e) {
            System.err.println(e);
            return false;
        } catch (IOException e) {
            System.err.println(e);
            return false;
        } finally {
            if (writer != null) {
                try {
                    writer.close();
                } catch (IOException ioe) {
                    return false;
                }
            }
        }
        return true;
    }

    public static void updateDictionary(RuleDictionary dict) {
        if (dict == null)
            return;

        List<SDKWarning> warnings = new ArrayList<SDKWarning>();
        try {
            dict.update(warnings);
            for (SDKWarning warning : warnings)
                System.out.println("warnings: " +
                                    warning.getLocalizedMessage());
        } catch (SDKException sdkEx) {
            sdkEx.printStackTrace();
        }
    }

    //You can call this method from your "Save" button

    public void saveDictionary() {

        RuleDictionary dict = this.getRuleDictModel().getRuleDictionary();

        if (dict != null) {
            if (dict.isModified())
                updateDictionary(dict);
            if (!dict.isTransactionInProgress())
                saveDictionary(dict, RULES_FILE1);
        }
    }

    //call validation method on the ruleDictModel to update Validation Panel

    public void validate() {
        if (this.ruleDictModel == null)
            return;
    }
}
```

```

        this.ruleDictModel.validate();
    }
}

```

6. Open the faces-config.xml file in the Overview mode and click the + button under Managed Beans to display the **Create Managed Bean** dialog box. Point to SomeBean.java by providing the Bean Name as someBean and the Scope as session as shown in [Figure 25-38](#).

Figure 25-38 Specifying the Bean Name and Scope



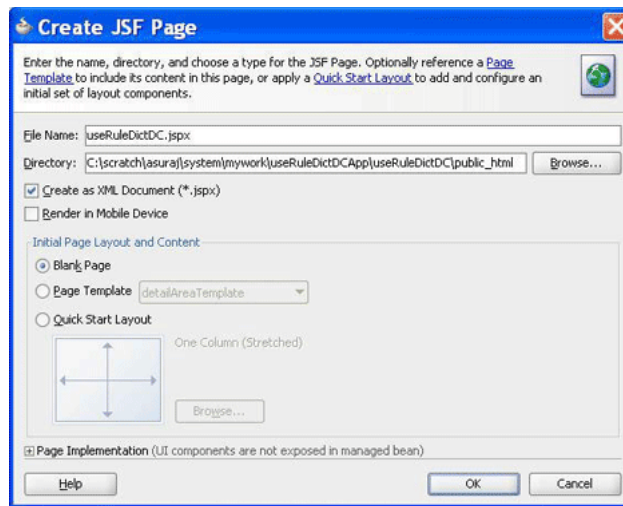
The ADF/JSF framework makes multiple calls to SomeBean.java to render the user interface. For example, `someBean.ruleDictModel` is called multiple times. So, it is better to create the `RuleDictModel` object once, cache it, and return it each time instead of re-creating it.

To create the .jspx file for the Rules Dictionary Editor Component tag:

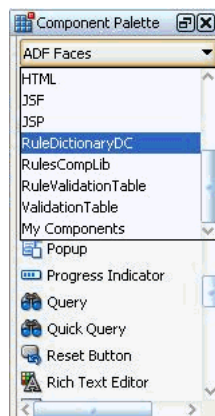
The next task is to create the .jspx file to include the Rules Dictionary Editor Component tag.

The steps are:

1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** to display the **New Gallery** dialog box.
3. In the **New Gallery** dialog box, select **JSF** under Web Tier from the Categories panel.
4. Select **JSF Page** under Items and click **OK** to display the **Create JSF Page** dialog box.
5. In the **Create JSF Page** dialog box, enter `useRuleDictDC.jsx` as the file name as shown in [Figure 25-39](#).

Figure 25–39 Specifying the Name of the JSF Page

RuleDictionaryDC in the component palette of Oracle JDeveloper is displayed as shown in Figure 25–40.

Figure 25–40 Rule Dictionary Editor Library in the Component Palette

This is because you have added Rules Dictionary Component when creating the sample application.

6. Select RuleDictionaryDC to view the ruleDictionaryDC tag. You can drag and drop the RuleDictionaryDC tag into the .jspx file. You can also add the RuleDictionaryDC tag in the .jspx file manually as shown:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:f="http://java.sun.com/jsf/core"
          xmlns:h="http://java.sun.com/jsf/html"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
          xmlns:rddc="http://xmlns.oracle.com/bpel/rules/dictionaryEditor">
  <jsp:directive.page contentType="text/html; charset=UTF-8"/>
  <f:view>
    <af:document id="d1" title="Sample Rule Dictionary App">
      <af:form id="f1">
        <af:panelStretchLayout id="psl1" inlineStyle="margin:15px;"
                              partialTriggers="cb2 cb3">
          <f:facet name="center">
```

```

<rddc:ruleDictionaryDC ruleDictModel="#{someBean.ruleDictModel}"
                        dtColumnPageSize="6" id="rddc1"
                        viewOnly="false" dateStyle="yyyy-MM-dd"
                        timeStyle="HH-mm-ss"
                        discloseRules="true"
                        showValidationPanel="true"/>
</f:facet>
<f:facet name="top">
  <af:panelGroupLayout id="pgl1" layout="horizontal">
    <af:commandButton text="Save Dict" id="cb2"
                      action="#{someBean.saveDictionary}"/>
    <af:spacer width="10" height="10" id="s1"/>
    <af:commandButton text="Validate" id="cb3"
                      action="#{someBean.validate}"/>
  </af:panelGroupLayout>
</f:facet>
</af:panelStretchLayout>
</af:form>
</af:document>
</f:view>
</jsp:root>

```

To refer to the oracle.rules and the oracle.soa.rules_dict_dc.webapp shared libraries:

After creating the .jspx file, you must refer to the `oracle.rules` and `oracle.soa.rules_editor_dc.webapp` shared libraries from the `weblogic-application.xml` file.

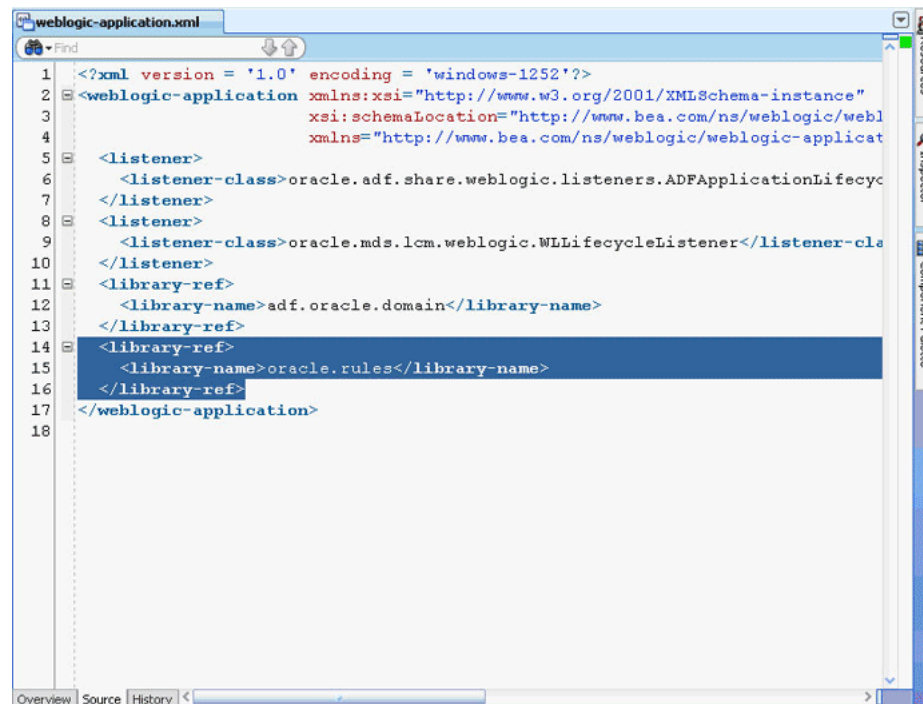
The steps are:

1. In Oracle JDeveloper, open the `weblogic-application.xml` file by browsing to Application Resources, then Descriptors, and then META-INF.
2. Add the following lines to refer to the `oracle.rules` shared library as shown in [Figure 25-41](#).

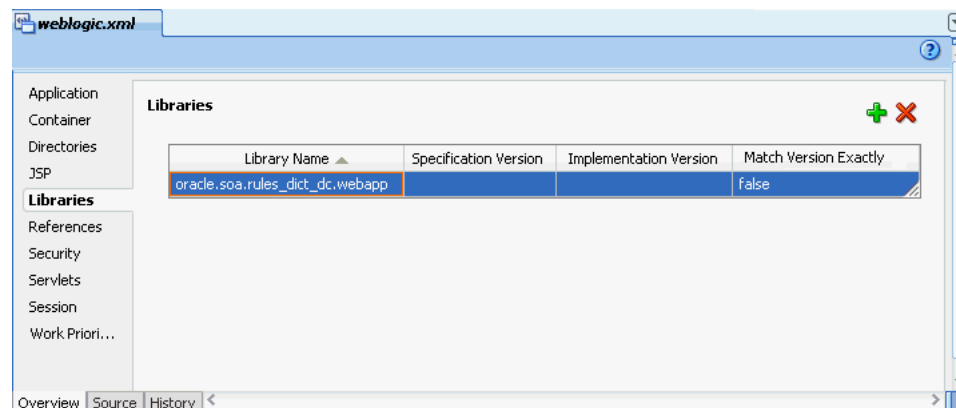
```

<library-ref>
<library-name>oracle.rules</library-name>
</library-ref>

```

Figure 25–41 Referring to the oracle.rules Shared Library

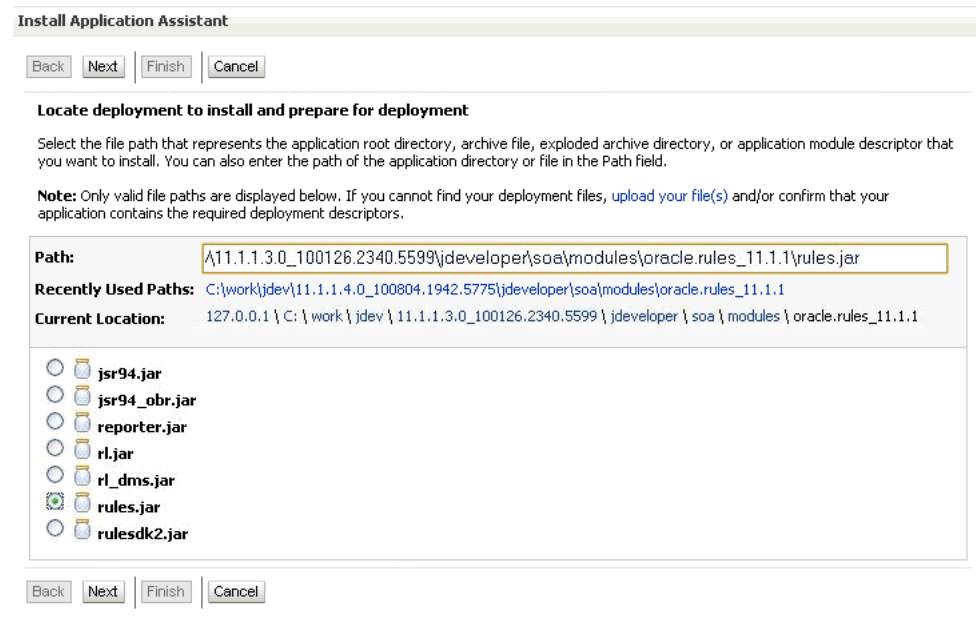
3. In Oracle JDeveloper,
 - a. From the **File** menu, select **New** to display the New Gallery dialog box.
 - b. In the New Gallery dialog box, select **Deployment Descriptors** under **General** from the Categories panel.
 - c. Select **Weblogic Deployment Descriptor** under **Items** and click **OK** to display the Create Weblogic Deployment Descriptor dialog box.
 - d. Select **weblogic.xml** from the list and click **Finish**.
 - e. In Oracle JDeveloper, in the Overview mode of `weblogic.xml`, select **Libraries** from the left panel and enter `oracle.soa.rules_dict_dc.webapp` as the library name as shown in Figure 25–42.

Figure 25–42 Adding the Rules Dictionary Component Library

4. Click **Save All**.

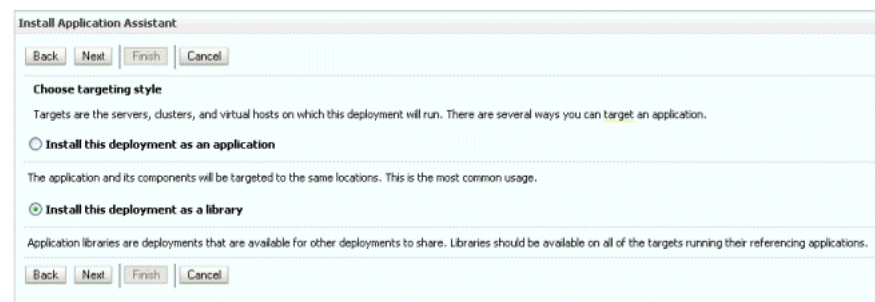
5. Deploy the `oracle.rules` shared library to the embedded Weblogic server:
 - a. Launch WLS console (<http://host:port/console/login/LoginForm.jsp>). Ensure that the Weblogic embedded server on Oracle JDeveloper is running.
 - b. Select **Deployments** and click **Install** to display the Install Application Assistant page.
 - c. Select `<JDEV_INSTALL>/jdeveloper/soa/modules/oracle.rules_11.1.1/rules.jar` and click **Finish** as shown in [Figure 25–43](#).

Figure 25–43 Deploying the oracle.rules Shared Library



6. Deploy the `oracle.soa.rules_dict_dc.webapp` shared library to the Weblogic server:
 - a. In the Weblogic console, select **Deployments** and click **Install** to display the Install Application Assistant page.
 - b. Select `<JDEV_INSTALL>/jdeveloper/soa/modules/oracle.soa.rules_dict_dc.webapp_11.1.1/oracle.soa.rules_dict_dc.webapp.war` and click **Next**.
 - c. Select **Install this deployment as a library** and click **Finish** as shown in [Figure 25–44](#).

Figure 25–44 Deploying oracle.soa.rules_editor_dc.webapp Shared Library



`oracle.soa.rules_dict_dc.webapp` is added to the list of deployments as shown in [Figure 25–45](#).

Figure 25–45 *oracle.soa.rules_dict_dc.webapp Added to the Deployment List*

☐	oracle.jrf.system.filter	Active	Library	100
☐	oracle.jsp.next(11.1.1,11.1.1)	Active	Library	100
☐	oracle.pwdgen(11.1.1,11.1.1.2.0)	Active	Library	100
☐	oracle.rules(11.1.1,11.1.1)	Active	Library	100
☐	oracle.soa.rules_dict_dc.webapp(11.1.1,11.1.1)	Active	Library	100
☐	oracle.wsm.seedpolicies(11.1.1,11.1.1)	Active	Library	100
☐	orai18n-adf(11,11.1.1.1.0)	Active	Library	100
☐	UIX(11,11.1.1.1.0)	Active	Library	100
☐	wsil-wls	Active	OK	Enterprise Application 5
☐	wsm-pm	Active	OK	Enterprise Application 5

Install Update Delete Start Stop

Showing 1 to 26 of 26 Previous Next

To run the sample Rules Dictionary Editor application:

The last task is running the sample application.

To run the sample application, from Oracle JDeveloper, right-click the `useRuleDictDC.jspx` file, and select **Run**. This starts the sample application on a Web browser as shown in [Figure 25–46](#).

Figure 25–46 *Running the Sample Rules Dictionary Editor Application*

Save Dict Validate

(x) Globals
Bucketsets
Rulesets
Ruleset_1
Ruleset_3

(x) Globals

Name	Description	Value	Bucketset	Type	Final
(x) temp	test	100	intRangeBS	int	☒
(x) Global_2	abc	'a'		char	☒
(x) Number Of Rows		Global_2+20	intBS	int	☒

Business Rule Validation - Log

Message	Dictionary Object	Property
RUL-05717: The identifier "test" is not valid here.	test/Ruleset_3/Rule_3/Customer/Test[1]/Test[1]/Ex Value	
RUL-05720: data type mismatch for test "!=": String != int	test/Ruleset_3/Rule_3/Customer/Test[2]/Expression Value	
RUL-05711: The expression cannot be blank.	test/Ruleset_3/Rule_4/Order/Test[2]/Expression[1]	
RUL-05711: The expression cannot be blank.	test/Ruleset_3/Rule_4/Order/Test[2]/Expression[2]	
RUL-05711: The expression cannot be blank.	test/Ruleset_3/DecisionTable_2/Condition[3]	
RUL-05712: The value "Target" of "Action" cannot be blank. Select a value.	test/Ruleset_3/DecisionTable_2/Action[2]	Target
RUL-05711: The expression cannot be blank.	test/This is a really long ruleset name/Rule_1/Pattern	
RUL-05711: The expression cannot be blank.	test/This is a really long ruleset name/Rule_1/Pattern	

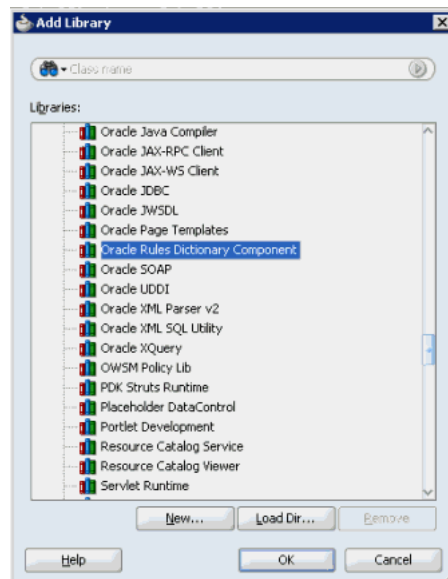
25.3.3 How to Deploy a Rules Dictionary Application to a Standalone Weblogic Server

When you are ready to deploy your application EAR file to the standalone Weblogic server, perform the following:

1. Launch the Weblogic server console (`http://host:port/console/login/LoginForm.jsp`) and ensure that `oracle.rules` is displayed in the deployments list.

2. Ensure that `oracle.soa.rules_dict_dc.webapp` is displayed in the deployments list. If this is not displayed, click **Install** and select the `<JDEV_INSTALL>/jdeveloper/soa/modules/oracle.soa.rules_dict_dc.webapp_11.1.1/oracle.soa.rules_dict_dc.webapp.war` file.
3. Open Oracle JDeveloper.
4. Right-click the project name in the Application Navigator and select **Project Properties**.
5. Select **Libraries and Classpath** from the left panel and click **Add Library**.
6. In the **Add Library** dialog box, select **Oracle Rules Dictionary Component** and click **OK** as shown in [Figure 25-47](#).

Figure 25-47 Adding the Oracle Rules Dictionary Component



This step enables you to refer to these libraries, but does not deploy these libraries by default. Therefore, the jar files are not included in your project war file.

7. In the project that has to be deployed (where you create the EAR file):
 - a. Add the following lines to the `weblogic-application.xml`:


```
<library-ref>
  <library-name>oracle.rules</library-name>
</library-ref>
```
 - b. Add the following lines to `weblogic.xml` in the project WAR file:


```
<library-ref>
  <library-name>oracle.soa.rules_dict_dc.webapp</library-name>
</library-ref>
```
 - c. Deploy the EAR file in the Weblogic server.

25.3.4 What You May Need to Know About the Supported Attributes of the Rules Dictionary Editor Component

This section lists the attributes that are supported by the Rules Dictionary Editor component.

Table 25–3 lists the supported attributes.

Table 25–3 Supported Rules Dictionary Editor Attributes

Name	Type	Required	Default Value	Supports EL?	Description
dateStyle	java.lang.String	no	Gets it from the locale	yes	If specified, the date style is used in all <code>inputDate</code> components, for example, <code>yyyy.MM.dd</code> .
decimalSeparator	java.lang.Character	no	Based on Locale	yes	Specifies the decimal separators. This is used in Number Formatting. If specified, this attribute overrides the decimal separator based on locale.
disableDFName	java.lang.Boolean	no	false	yes	If true, the Decision Function Name in the Decision Function editor window is disabled.
disableInputOps	java.lang.Boolean	no	false	yes	Disables add, edit, and delete operations for the Inputs table in the Decision Function editor window.
disableOutputOps	java.lang.Boolean	no	false	yes	Disables add, edit, and delete operations for the Outputs table in the Decision Function editor window.
disableRuleSetName	java.lang.Boolean	no	false	yes	If true, the editable ruleset name is disabled. This attribute is used only when <code>displayRuleSetName</code> is set to true.
discloseRules	java.lang.Boolean	no	false	yes	If true, all the rules in the ruleset are disclosed or expanded. If false, all the rules are collapsed.
displayAddDF	java.lang.Boolean	no	true	yes	Displays the Add Decision Function button.
displayDeleteDF	java.lang.Boolean	no	true	yes	Displays the Delete Decision Function button.
displayRuleSetName	java.lang.Boolean	no	true	yes	Displays the editable ruleset name by default. You can choose to hide this by setting this to false.

Table 25–3 (Cont.) Supported Rules Dictionary Editor Attributes

Name	Type	Required	Default Value	Supports EL?	Description
displayWSCheck	java.lang.Boolean	no	true	yes	If true, the Invoke as rule service check box in the Decision Function editor window is displayed.
displayWSName	java.lang.Boolean	no	true	yes	If true, the decision service name is displayed in the Decision Function editor window. Note that the service name is relevant only when Invoke as rule service is selected.
dtColumnPageSize	java.lang.Integer	no	5	yes	Number of columns to be displayed at a time in the decision table. This works only when rules are columnar.
dtHeight	java.lang.Integer	no	16	yes	Number of rows to be displayed at a time in the decision table. A scroll bar is displayed if the number of rows increases over the specified height.
groupingSeparator	java.lang.Character	no	Based on Locale	yes	Specifies the grouping separators. This is used in Number Formatting. If specified, this attribute overrides the grouping separator based on locale.
locale	java.util.Locale	no	Locale.getDefault()	yes	Used for Localization
ruleDictModel	oracle.bpel.ruledictionarydc.model.interfaces.RuleDictionaryInterface	yes	-	Only EL	Wrapper around the Rules SDK Dictionary object. You can use the RuleDictionaryModel object supplied as part of the Rules Dictionary Editor Component jar file (adflibRuleDictionaryDC.jar).
rulesPageSize	java.lang.Integer	no	5	yes	Specifies the number of rules to be displayed in a page. It is used in IF/THEN rules pagination.

Table 25–3 (Cont.) Supported Rules Dictionary Editor Attributes

Name	Type	Required	Default Value	Supports EL?	Description
selectedTab	java.lang.String	no	-	yes	Switches to the specified tab name (either GLOBALS, BUCKETSETS, DESC_FUNCS or the ruleset name).
showDTButtons	java.lang.Boolean	no	true	yes	Displays the add and delete Decision Table buttons.
showValidationPanel	java.lang.Boolean	no	true	yes	Displays the validation panel by default. You can choose to hide this by setting this to false.
timeStyle	java.lang.String	no	Gets it from the locale	yes	If specified, the time style is used in all inputDate components, for example, HH:mm:ss.
timezone	java.util.TimeZone	no	TimeZone.getDefault()	yes	Used for Localization
viewOnly	java.lang.Boolean	no	true	yes	If true, in the "viewOnly" mode, you can view the existing dictionary data, but you cannot edit the data. If false, which is the "edit" mode, you can edit existing the dictionary data.

25.4 Using the Oracle Business Rules Dictionary Task Flow

This section discusses the Oracle Business Rules Dictionary Editor task flow. It also provides information on how to create and run an application using the Rules Dictionary Editor task flow, and then deploy the application.

25.4.1 Introduction to the Oracle Business Rules Dictionary Task Flow

The Rules Dictionary Editor Task Flow is basically a wrapper around the Rules Dictionary Editor declarative component. The task flow is used in ADF-based Web applications that require a task flow instead of a declarative component. For more information on Rules Dictionary Editor Component, see [Section 25.3, "Using the Oracle Business Rules Dictionary Editor Declarative Component."](#)

25.4.2 How to Create and Run a Sample Application By Using the Rules Dictionary Editor Task Flow

This section lists the steps for creating and running a sample application by using the Rules Dictionary Editor task flow.

The prerequisites for using the Rules Dictionary Editor task flow to create ADF-based Web applications is having a running installation of SOA Suite and Oracle JDeveloper on your computer.

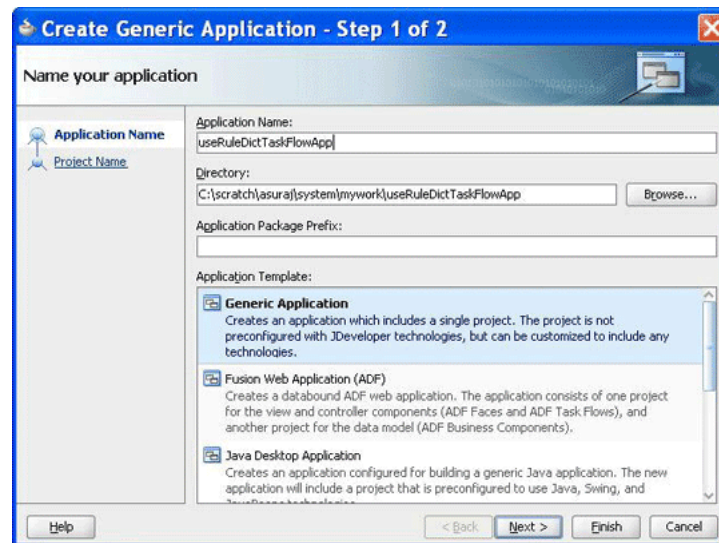
To create a sample application by using the Rules Dictionary Editor task flow:

The first task is to create a sample application.

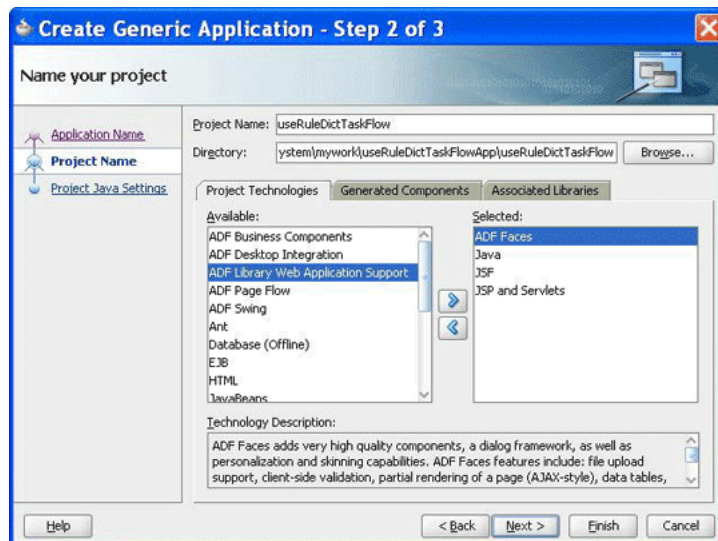
The steps are:

1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** and then **Generic Application** to create an application.
3. Enter a name for the application in the **Application Name** field, for example, `useRuleDictTaskFlowApp`, and click **Next** as shown in [Figure 25–48](#).

Figure 25–48 *Creating a Generic Task Flow Application*



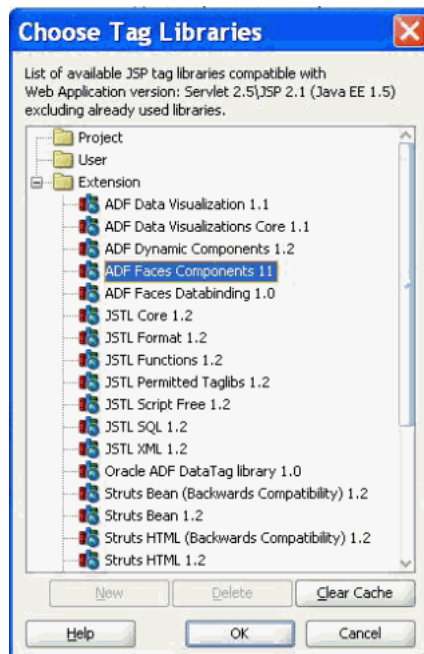
4. Enter `useRuleDictTaskFlow` in the **Project Name** field and ensure that **ADF Faces** is selected in the **Project Technologies** tab, as shown in [Figure 25–49](#). Click **Finish** to create the project.

Figure 25–49 Creating a Task Flow Project

5. Right-click the **useRuleDictTaskFlow** project in the Application Navigator of Oracle JDeveloper, and select **Project Properties** to display the **Project Properties** dialog box.

In the Project Properties dialog box:

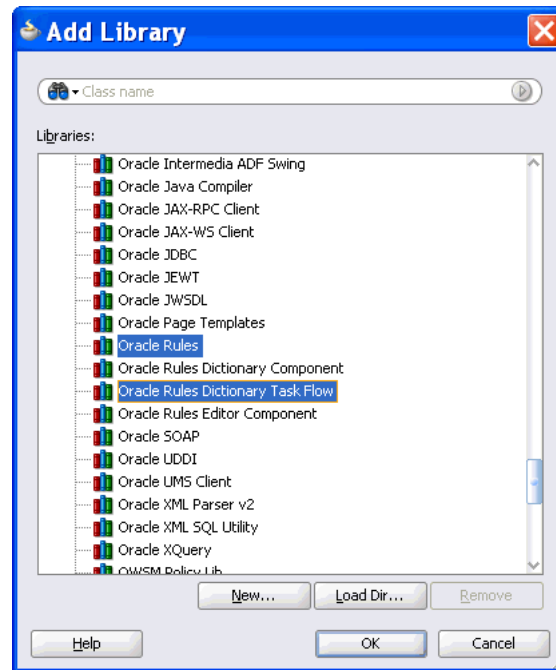
- a. Select **JSP Tag Libraries** from the left panel.
- b. Click **Add** and select **ADF Faces Components** from the extension list in the Choose Tag Libraries dialog box, and click **OK** as shown in [Figure 25–50](#).

Figure 25–50 Choosing Tag Libraries for the Task Flow Application

- c. Select **Libraries and Classpath** from the left panel and click **Add Library** to display the **Add Library** dialog box.

- d. Select **Oracle Rules** and then **Oracle Rules Dictionary Task Flow** in the Libraries list and click **OK** as shown in [Figure 25–51](#). This adds the Rules SDK and the Rules Dictionary Task Flow JARs to the project.

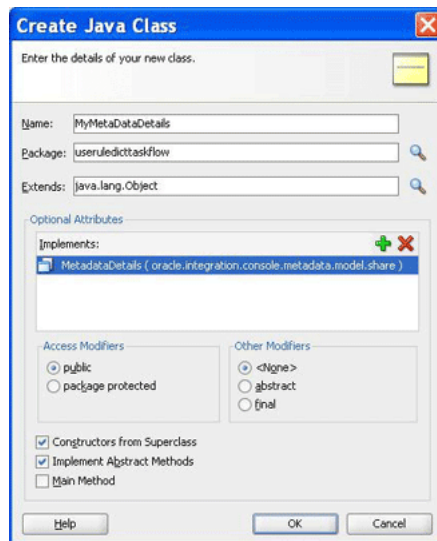
Figure 25–51 Adding the Rules SDK and Rules Dictionary Task Flow



- e. Click **OK** to close the Project Properties dialog box.
6. Click **Save All** from the Oracle JDeveloper **File** menu to save the project.
7. Create a Java class that implements the `oracle.integration.console.metadata.model.share.MetadataDetails` interface, which is defined in `soaComposerTemplates.jar`. For more information on the `MetadataDetails` interface, see [Section I.1, "The MetadataDetails Interface."](#)

The steps are:

- a. Open Oracle JDeveloper.
- b. From the **File** menu, select **New** to display the **New Gallery** dialog box.
- c. In the **New Gallery** dialog box, select **Java** under **General** from the Categories panel. Ensure that **Java Class** under **Items** is selected and click **OK** to display the **Create Java Class** dialog box.
- d. Enter the name of the Java class, for example `MyMetaDataDetails`, add the `MetadataDetails` interface in the **Implements** box under **Optional Attributes**, and click **OK** to create the Java class in your project as shown in [Figure 25–52](#).

Figure 25–52 Creating a Java Class That Implements the MetadataDetails Interface

The following is a sample of the content of the MyMetadataDetails.java file:

```
package useruledicttaskflow;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.io.Writer;

import java.net.MalformedURLException;
import java.net.URL;

import oracle.integration.console.metadata.model.share.MetadataDetails;
import oracle.integration.console.metadata.model.share.RelatedMetadataPath;

public class MyMetadataDetails implements MetadataDetails {
    public MyMetadataDetails() {
        super();
    }

    private static final String RULES_FILE1 =
        "file:///<path of Rules file>";

    public String getDocument() {
        URL url = null;
        try {
            url = new URL(RULES_FILE1);
            return readFile(url);
        } catch (IOException e) {
            System.err.println(e);
        }
        return "";
    }
}
```

```

public void setDocument(String string) {
    URL url = null;

    try {
        url = new URL(RULES_FILE1);
    } catch (MalformedURLException e) {
        System.err.println(e);
        return;
    }
    Writer writer = null;
    try {
        //os = new FileWriter(url.getPath());
        writer =
            new OutputStreamWriter(new FileOutputStream(url.getPath()),
                "UTF-8");
    } catch (FileNotFoundException e) {
        System.err.println(e);
        return;
    } catch (IOException e) {
        System.err.println(e);
        return;
    }
    try {
        writer.write(string);
    } catch (IOException e) {
        System.err.println(e);
    } finally {
        if (writer != null) {
            try {
                writer.close();
            } catch (IOException ioe) {
                System.err.println(ioe);
            }
        }
    }
}

private String readFile(URL dictURL) {
    InputStream is;
    try {
        is = dictURL.openStream();
    } catch (IOException e) {
        System.err.println(e);
        return "";
    }
    BufferedReader reader;
    try {
        reader = new BufferedReader(new InputStreamReader(is, "UTF-8"));
    } catch (UnsupportedEncodingException e) {
        System.err.println(e);
        return "";
    }
    String line = null;
    StringBuilder stringBuilder = new StringBuilder();
    String ls = System.getProperty("line.separator");
    try {
        while ((line = reader.readLine()) != null) {
            stringBuilder.append(line);
            stringBuilder.append(ls);
        }
    }
}

```

```
        } catch (IOException e) {
            System.err.println(e);
            return "";
        } finally {
            try {
                reader.close();
            } catch (IOException e) {
                System.err.println(e);
            }
        }
    }
    return stringBuilder.toString();
}

public String getRelatedDocument(RelatedMetadataPath relatedMetadataPath) {
    String currPath =
        RULES_FILE1.substring(0, RULES_FILE1.indexOf("oracle/rules"));
    String relatedDoc =
        currPath + "oracle/rules/" + relatedMetadataPath.getValue();

    URL url = null;
    try {
        url = new URL(relatedDoc);
        return readFile(url);
    } catch (IOException e) {
        System.err.println(e);
    }
    return "";
}
}
```

8. Create a Java class called `MyNLSPreferences` that implements the `oracle.integration.console.metadata.model.share.NLSPreference` s interface, which is defined in `soaComposerTemplates.jar`.

For more information about the NLS Preferences interface, see [Section I.2, "The NLSPreferences Interface."](#)

The following is a sample of `MyNLSPreferences.java` that implements the `NLSPreferences` interface:

```
package useruledicctaskflow;

import java.util.Locale;
import java.util.TimeZone;

import oracle.integration.console.metadata.model.share.NLSPreferences;

public class MyNLSPreferences implements NLSPreferences {
    private static final String DATE_STYLE = "yyyy-MM-dd";
    private static final String TIME_STYLE = "HH-mm-ss";

    public MyNLSPreferences() {
        super();
    }

    public Locale getLocale() {
        return Locale.getDefault();
    }

    public TimeZone getTimeZone() {
        return TimeZone.getTimeZone("America/Los_Angeles");
    }
}
```

```

    }

    public String getDateFormat() {
        return DATE_STYLE;
    }

    public String getTimeFormat() {
        return TIME_STYLE;
    }
}

```

9. Create a Managed Bean called `MyBean.java` to return the implementation of `MetadataDetails` and `NLSPreferences`. It also returns the `oracle.integration.console.metadata.model.share.MetadataDetailsMode` object and provides event handlers such as `toggleMode()`, `saveDictionary()`, `saveNoValidateDictionary()`, and `validate()`.

The following is a sample of the `MyBean.java` file:

```

package useruledicctaskflow;

import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;

import javax.faces.context.FacesContext;
import javax.faces.event.PhaseId;

import oracle.adf.view.rich.component.rich.fragment.RichRegion;

import oracle.integration.console.metadata.model.share.MetadataDetails;
import oracle.integration.console.metadata.model.share.MetadataDetailsMode;
import oracle.integration.console.metadata.model.share.NLSPreferences;

public class MyBean {
    private MyMetadataDetails details = null;
    private MetadataDetailsMode mode = MetadataDetailsMode.VIEW;
    private RichRegion regionComp;
    private NLSPreferences nlsPrefs;

    public MyBean() {
        super();
    }

    public MetadataDetails getMetaDataDetails() {
        if (details != null)
            return details;

        details = new MyMetadataDetails();
        return details;
    }

    public MetadataDetailsMode getDetailsMode() {
        return mode;
    }

    public void toggleMode() {
        if (mode.equals(MetadataDetailsMode.EDIT))
            mode = MetadataDetailsMode.VIEW;
        else
            mode = MetadataDetailsMode.EDIT;
    }
}

```

```
    }

    public void saveDictionary() {
        if (regionComp == null)
            return;
        FacesContext fc = FacesContext.getCurrentInstance();
        ExpressionFactory ef = fc.getApplication().getExpressionFactory();
        ELContext elc = fc.getELContext();
        MethodExpression me =
            ef.createMethodExpression(elc, "doMetadataUpdate", String.class,
                                     new Class[] { });
        regionComp.queueActionEventInRegion(me, null, null, false, -1, -1,
                                           PhaseId.ANY_PHASE);
    }

    public void saveNoValidateDictionary() {
        if (regionComp == null)
            return;
        FacesContext fc = FacesContext.getCurrentInstance();
        ExpressionFactory ef = fc.getApplication().getExpressionFactory();
        ELContext elc = fc.getELContext();
        MethodExpression me =
            ef.createMethodExpression(elc, "doNoValidateMetadataUpdate",
                                     String.class, new Class[] { });
        regionComp.queueActionEventInRegion(me, null, null, false, -1, -1,
                                           PhaseId.ANY_PHASE);
    }

    public void validate() {
        if (regionComp == null)
            return;
        FacesContext fc = FacesContext.getCurrentInstance();
        ExpressionFactory ef = fc.getApplication().getExpressionFactory();
        ELContext elc = fc.getELContext();
        MethodExpression me =
            ef.createMethodExpression(elc, "doValidate", String.class,
                                     new Class[] { });
        regionComp.queueActionEventInRegion(me, null, null, false, -1, -1,
                                           PhaseId.ANY_PHASE);
    }

    public void setRegionComp(RichRegion regionComp) {
        this.regionComp = regionComp;
    }

    public RichRegion getRegionComp() {
        return regionComp;
    }

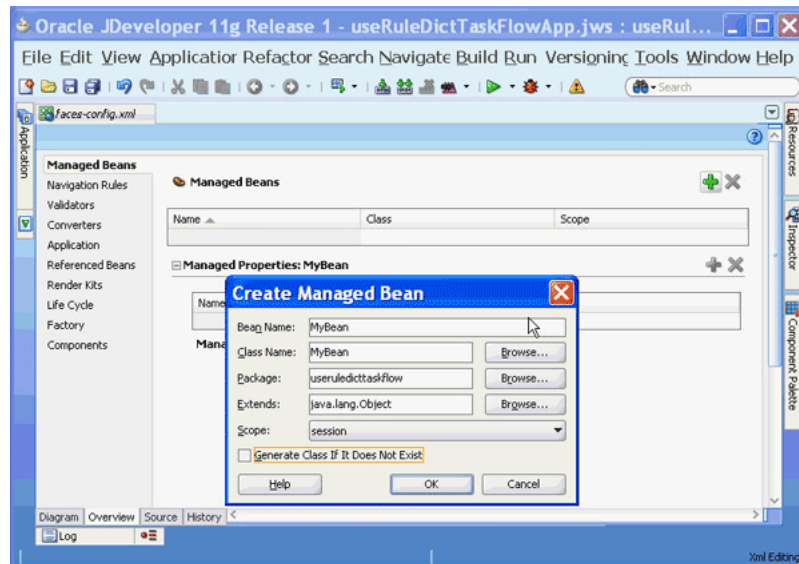
    public NLSPreferences getNlsPrefs() {
        if (nlsPrefs != null)
            return nlsPrefs;

        nlsPrefs = new MyNLSPreferences();
        return nlsPrefs;
    }
}
```

10. Open the `faces-config.xml` file in the Overview mode and click the + button under **Managed Beans** to display the **Create Managed Bean** dialog box. Point to

MyBean.java by providing the Bean Name as MyBean and the Scope as session as shown in [Figure 25–53](#).

Figure 25–53 Specifying the Bean Name and Scope in the Task Flow Application



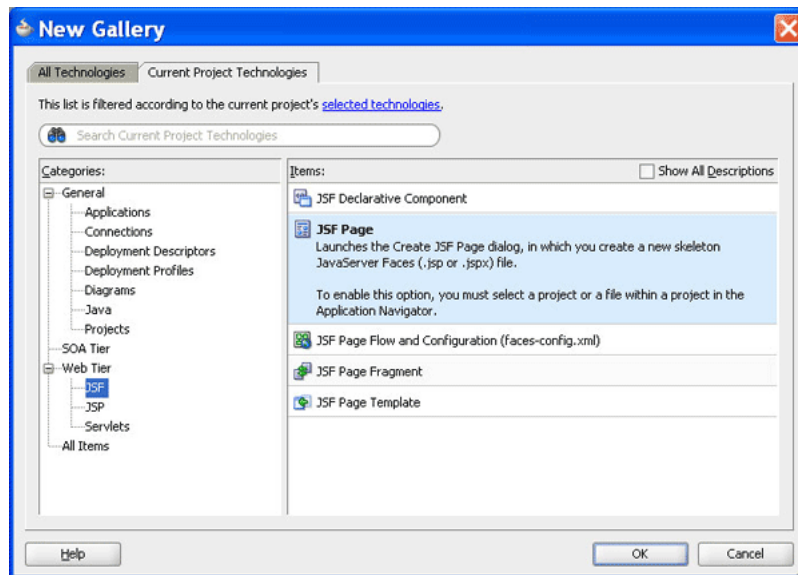
To add a Rules Dictionary Editor task flow in a .jspx file:

The next task is to create the .jspx file to include the Rules Dictionary Editor component tag.

The steps are:

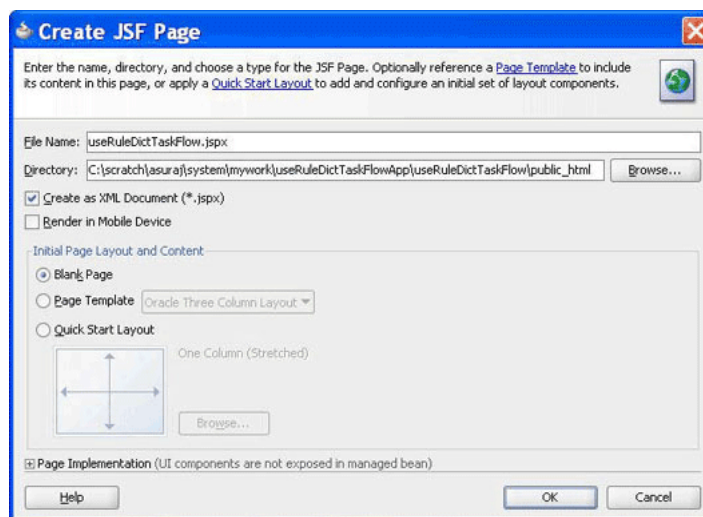
1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** to display the **New Gallery** dialog box.
3. In the **New Gallery** dialog box, select **JSF** under Web Tier from the Categories panel.
4. Select **JSF Page** under Items and click **OK** to display the **Create JSF Page** dialog box as shown in [Figure 25–54](#).

Figure 25–54 Creating the JSF Page File to Include the Rules Dictionary Editor Task Flow

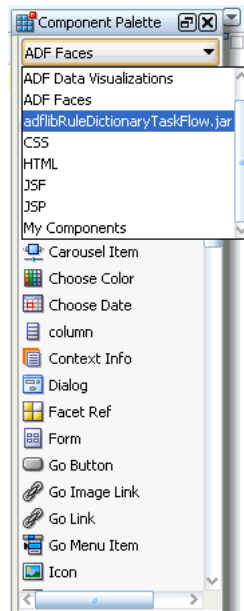


5. In the **Create JSF Page** dialog box, enter `useRuleDictTaskFlow.jspx` as the file name as shown in [Figure 25–55](#).

Figure 25–55 Specifying the Name of the JSF Page for the Task Flow

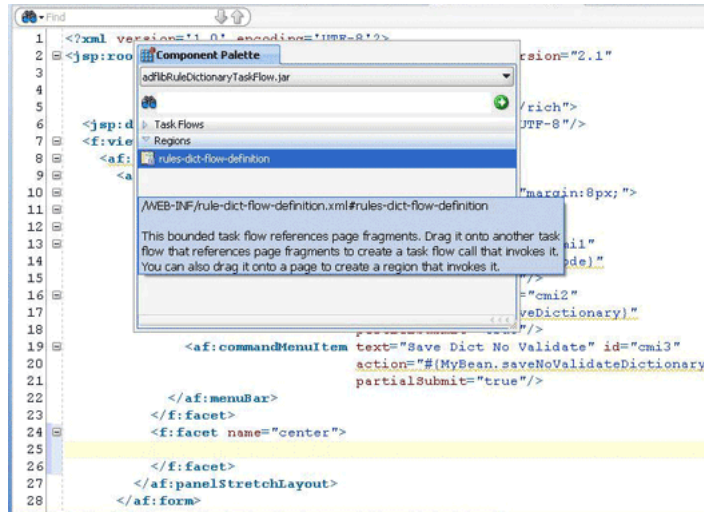


`adflibRuleDictionaryTaskFlow.jar` is displayed in the component palette of Oracle JDeveloper as shown in [Figure 25–56](#).

Figure 25–56 Rules Dictionary Task Flow JAR in the Component Palette

This is because you have added the Oracle Rules Dictionary Task Flow shared library when creating the sample application.

6. Select `adflibRuleDictionaryTaskFlow.jar` to make `rule-dict-flow-definition` to be available under Regions in the component palette. You can drag and drop the `rule-dict-flow-definition` region into the `.jspx` file as shown in [Figure 25–57](#), and specify all the required parameters.

Figure 25–57 Dragging and Dropping the Region

The following is a sample of the `useRuleDictTaskFlow.jsx` file with the task flow added:

```
<f:view>
  <af:document id="d1">
    <af:form id="f1">
      <af:panelStretchLayout id="ps11" inlineStyle="margin:8px;">
        <f:facet name="top">
```

```

<af:menuBar id="mb1">
  <af:commandMenuItem text="Toggle Mode" id="cmi1"
    action="{MyBean.toggleMode}"
    partialSubmit="true"/>
  <af:commandMenuItem text="Save Dict" id="cmi2"
    action="{MyBean.saveDictionary}"
    partialSubmit="true"/>
  <af:commandMenuItem text="Save Dict No Validate" id="cmi3"
    action="{MyBean.saveNoValidateDictionary}"
    partialSubmit="true"/>
  <af:commandMenuItem text="Validate" id="cmi4"
    action="{MyBean.validate}"
    partialSubmit="true"/>
</af:menuBar>
</f:facet>
<f:facet name="center">
  <af:region value="{bindings.rulesdictflowdefinition1.regionModel}"
    id="r2" binding="{MyBean.regionComp}"
    partialTriggers="::cmi1 ::cmi2 ::cmi3 ::cmi4"/>
</f:facet>
</af:panelStretchLayout>
</af:form>
</af:document>
</f:view>

```

In the preceding sample, you can find code snippets for rendering the following buttons to the page:

- **Toggle Mode:** Enables switching between Read-Only and Editable modes of SOA Composer
- **Save Dict:** Enables saving the dictionary (with or without validation)

To edit the pagedef.xml file:

After you add the task flow to the .jspx file, you must edit the useRuleDictTaskFlowPageDef.xml file. The pagedef.xml file is created when you drop the Rules Dictionary task flow into the .jspx page.

The following is a sample of the pagedef.xml file along with all the parameters that must be passed to the Rules Dictionary task flow:

```

<?xml version="1.0" encoding="UTF-8" ?>
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel"
  version="11.1.1.55.99" id="useRuleDictTaskFlowPageDef"
  Package="useruledicttaskflow.pageDefs">
  <parameters/>
  <executables>
    <variableIterator id="variables"/>
    <taskFlow id="rulesdictflowdefinition1"
      taskFlowId= "/WEB-INF/rule-dict-flow-definition.xml#rules-dict-flow-definition"
      activation="deferred"
      xmlns="http://xmlns.oracle.com/adf/controller/binding">
    </parameters>
    <parameter id="details" value="{MyBean.metadataDetails}"
      xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
    <parameter id="mode" value="{MyBean.detailsMode}"
      xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
    <parameter id="dtHeight" value="10"
      xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
    <parameter id="selectedTab" value="Ruleset_1"
      xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  </executables>
</pageDefinition>

```

```

<parameter id="dtColumnPageSize" value="6"
  xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="nlsPrefs" value="#{MyBean.nlsPrefs}"
  xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="discloseRules" value="true"
  xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
</parameters>
</taskFlow>
</executables>
<bindings/>
</pageDefinition>

```

To refer to the oracle.rules and the oracle.soa.rules_dict_dc.webapp shared libraries:

The next task is to refer to the `oracle.rules` and `oracle.soa.rules_dict_dc.webapp` shared libraries from the `weblogic-application.xml` file.

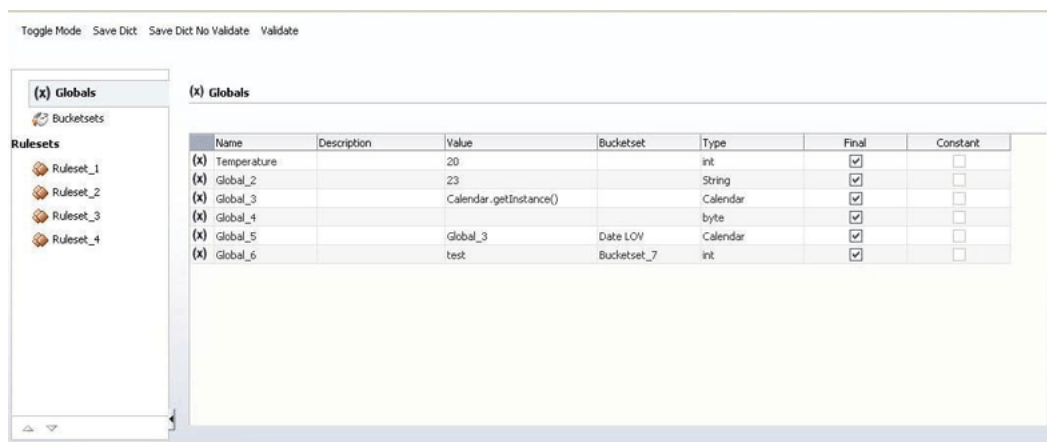
For more information on referring to the shared libraries, see [Section 25.3.2, "How to Create and Run a Sample Application by Using the Rules Dictionary Editor Component."](#)

To run the sample task flow application:

The last task is running the sample application in the embedded server.

To run the sample application, from Oracle JDeveloper, right-click the `useRulesDictTaskFlow.jspx` file, and select **Run**. This starts the sample application on a Web browser as shown in [Figure 25–46](#).

Figure 25–58 Running the Sample Rules Dictionary Editor Task Flow Application



25.4.3 How to Deploy a Rules Dictionary Editor Task Flow Application to a Standalone Weblogic Server

When you are ready to deploy your application EAR file to the standalone Weblogic server, perform the following:

1. Launch the Weblogic server console (`http://host:port/console/login/LoginForm.jsp`) and ensure that `oracle.rules` is displayed in the deployments list.

2. Ensure that `oracle.soa.rules_dict_dc.webapp` is displayed in the deployments list. If this is not displayed, click **Install** and select the `<JDEV_INSTALL>/jdeveloper/soa/modules/oracle.soa.rules_dict_dc.webapp_11.1.1/oracle.soa.rules_dict_dc.webapp.war` file.
3. In the project that has to be deployed (where you create the EAR file):
 - a. Add the following lines to the `weblogic-application.xml`:

```
<library-ref>
  <library-name>oracle.rules</library-name>
</library-ref>
```
 - b. Add the following lines to `weblogic.xml` in the project WAR file:

```
<library-ref>
  <library-name>oracle.soa.rules_dict_dc.webapp</library-name>
</library-ref>
```
 - c. Deploy the EAR file in the Weblogic server.

25.5 Localizing the ADF-Based Web Application

You can localize an application that is created using the Rules Editor component, Rules Dictionary Editor component, or the Rules Dictionary Editor task flow.

The steps are:

1. Modify `faces-config.xml` in the project that uses the Rules Editor component. The `faces-config.xml` file must have the following code within the `<application>` tag to support the available resource bundles:

```
<locale-config>
  <default-locale>en</default-locale>
  <supported-locale>en</supported-locale>
  <supported-locale>ar</supported-locale>
  <supported-locale>cs</supported-locale>
  <supported-locale>da</supported-locale>
  <supported-locale>de</supported-locale>
  <supported-locale>el</supported-locale>
  <supported-locale>es</supported-locale>
  <supported-locale>fi</supported-locale>
  <supported-locale>fr</supported-locale>
  <supported-locale>hu</supported-locale>
  <supported-locale>it</supported-locale>
  <supported-locale>iw</supported-locale>
  <supported-locale>ja</supported-locale>
  <supported-locale>ko</supported-locale>
  <supported-locale>nl</supported-locale>
  <supported-locale>no</supported-locale>
  <supported-locale>pl</supported-locale>
  <supported-locale>pt-BR</supported-locale>
  <supported-locale>pt</supported-locale>
  <supported-locale>ro</supported-locale>
  <supported-locale>ru</supported-locale>
  <supported-locale>sk</supported-locale>
  <supported-locale>sv</supported-locale>
  <supported-locale>th</supported-locale>
  <supported-locale>tr</supported-locale>
  <supported-locale>zh-CN</supported-locale>
  <supported-locale>zh-TW</supported-locale>
</locale-config>
```

2. Change the browser language to the locale of your choice.
3. You can override the locale provided by the browser and display the user interface in a specific locale. This is done by passing that locale as an attribute to the component and modifying the `f:view` tag in the application using the component as shown:

```
<f:view locale="#{someBean.locale}">
```

The `locale` specified here should be the same as the one passed to the component using the `locale` attribute.

Part V

Using the Human Workflow Service Component

This part describes how to use the human workflow service component.

This part contains the following chapters:

- [Chapter 26, "Getting Started with Human Workflow"](#)
- [Chapter 27, "Designing Human Tasks"](#)
- [Chapter 28, "Designing Task Forms for Human Tasks"](#)
- [Chapter 29, "Using Oracle BPM Worklist"](#)
- [Chapter 30, "Building a Custom Worklist Client"](#)
- [Chapter 31, "Introduction to Human Workflow Services"](#)
- [Chapter 32, "Integrating Microsoft Excel with a Human Task"](#)
- [Chapter 33, "Configuring Task List Portlets"](#)

Getting Started with Human Workflow

This chapter introduces human workflow concepts, features, and architecture. Use cases for human workflow are provided. Instructions for designing your first workflow from start to finish are also provided.

This chapter includes the following sections:

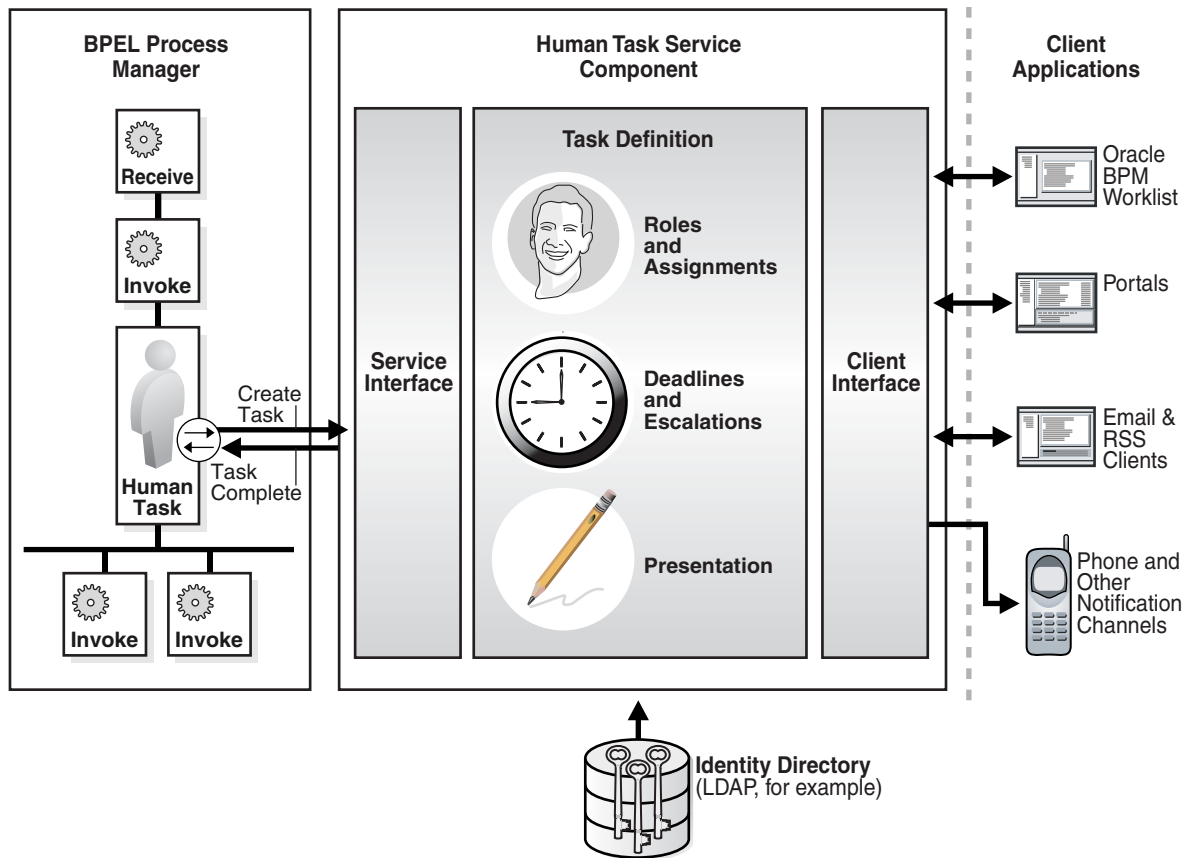
- [Section 26.1, "Introduction to Human Workflow"](#)
- [Section 26.2, "Introduction to Human Workflow Concepts"](#)
- [Section 26.3, "Introduction to Human Workflow Features"](#)
- [Section 26.4, "Introduction to Human Workflow Architecture"](#)

26.1 Introduction to Human Workflow

Many end-to-end business processes require human interactions with the process. For example, humans may be needed for approvals, exception management, or performing activities required to advance the business process. The human workflow component provides the following features:

- Human interactions with processes, including assignment and routing of tasks to the correct users or groups
- Deadlines, escalations, notifications, and other features required for ensuring the timely performance of a task (human activity)
- Presentation of tasks to end users through a variety of mechanisms, including a worklist application (Oracle BPM Worklist)
- Organization, filtering, prioritization, and other features required for end users to productively perform their tasks
- Reports, reassignments, load balancing, and other features required by supervisors and business owners to manage the performance of tasks

[Figure 26-1](#) provides an overview of human workflow.

Figure 26–1 Human Workflow

In Figure 26–1, the following actions occur:

- A BPEL process invokes a special activity of the human task type when it needs a human to perform a task.
- This creates a task in the human task service component. The process waits for the task to complete. It is also possible for the process to watch for other callbacks from the task and react to them.
- There is metadata associated with the task that is used by the human task service component to manage the lifecycle of the task. This includes specification of the following:
 - Who performs the task. If multiple people are required to perform the task, what is the order?
 - Who are the other stakeholders?
 - When must the task be completed?
 - How do users perform the task, what information is presented to them, what are they expected to provide, and what actions can they take?
- The human task service component uses an identity directory, such as LDAP, to determine people's roles and privileges.
- The human task service component presents tasks to users through a variety of channels, including the following:

- Oracle BPM Worklist, a role-based application that supports the concept of supervisors and process owners, and provides functionality for finding, organizing, managing, and performing tasks.
- Worklist functionality is also available as portlets that can be exposed in an enterprise portal.
- Notifications can be sent by email, phone, SMS, and other channels. Email notifications can be actionable, enabling users to perform actions on the task from within the email client without connecting to Oracle BPM Worklist or Oracle WebLogic Server.

For information about portlets, see [Chapter 33, "Configuring Task List Portlets."](#)

26.2 Introduction to Human Workflow Concepts

This section introduces you to key human workflow design time and runtime concepts. This section also provides an overview of the three main stages of human workflow design.

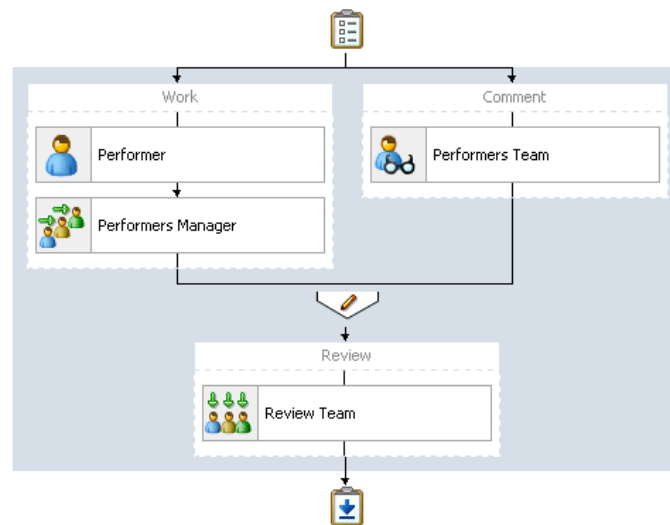
26.2.1 Introduction to Design and Runtime Concepts

Before designing a human task, it is important to understand the design and runtime concepts. A typical task consists of a subject, priority, task participants, task parameters or data, deadlines, notifications or reminders, and task forms. This section provides an overview of key concepts.

Note: Human workflow design-time tasks are performed in a graphical editor known as the Human Task Editor. The tutorial in [Section 26.3.2, "Designing a Human Task from Start to Finish"](#) describes how to use this editor.

26.2.1.1 Task Assignment and Routing

Human workflow supports declarative assignment and routing of tasks. In the simplest case, a task is assigned to a single participant (user or group). However, there are many situations in which more detailed task assignment and routing is necessary (for example, when a task must be approved by a management chain or worked and voted on by a set of people in parallel, as shown in [Figure 26–2](#)). Human workflow provides declarative, pattern-based support for such scenarios.

Figure 26–2 Participants in a Task

26.2.1.1.1 Participant A participant is a user or set of users in the assignment and routing policy definition. In [Figure 26–2](#), each block with an icon representing people is a participant.

26.2.1.1.2 Participant Type In simple cases, a participant maps to a user, group, or role. However, as discussed in [Section 26.2.1.1, "Task Assignment and Routing,"](#) workflow supports declarative patterns for common routing scenarios such as management chain and group vote. The following participant types are available:

- **Single approver**

This is the simple case where a participant maps to a user, group, or role.

For example, a vacation request is assigned to a manager. The manager must act on the request task three days before the vacation starts. If the manager formally approves or rejects the request, the employee is notified with the decision. If the manager does not act on the task, the request is treated as rejected. Notification actions similar to the formal rejection are taken.

- **Parallel**

This participant indicates that a set of people must work in parallel. This pattern is commonly used for voting.

For example, multiple users in a hiring situation must vote to hire or reject an applicant. You specify the voting percentage that is needed for the outcome to take effect, such as a majority vote or a unanimous vote.

- **Serial**

This participant indicates that a set of users must work in sequence. While working in sequence can be specified in the routing policy by using multiple participants in sequence, this pattern is useful when the set of people is dynamic. The most common scenario for this is management chain escalation, which is done by specifying that the list is based on a management chain within the specification of this pattern.

- **FYI (For Your Information)**

This participant also maps to a single user, group, or role, just as in single approver. However, this pattern indicates that the participant just receives a notification task and the business process does not wait for the participant's response. FYI participants cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.

For example, a regional sales office is notified that a candidate for employment has been approved for hire by the regional manager and their candidacy is being passed onto the state wide manager for approval or rejection. FYIs cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.

For more information, see [Section 27.3.6, "How to Assign Task Participants."](#)

26.2.1.1.3 Participant Assignment A task is work that must be done by a user. When you create a task, you assign humans to participate in and act upon the task. Participants can perform actions upon tasks during runtime from Oracle BPM Worklist, such as approving a vacation request, rejecting a purchase order, providing feedback on a help desk request, or some other action. There are three types of participants:

- Users

You can assign individual users to act upon tasks. For example, you may assign users `j_london` or `j_stein` to a particular task. Users are defined in an identity store configured with the SOA Infrastructure. These users can be in the embedded LDAP of Oracle WebLogic Server, Oracle Internet Directory, or a third party LDAP directory.

- Groups

You can assign groups to act upon tasks. Groups contain individual users who can claim and act upon a task. For example, users `j_cooper` and `f_kafka` may be members of the group `LoanAgentGroup` that you assign to act upon the task.

As with users, groups are defined in the identity store of the SOA Infrastructure.

- Application roles

You can assign users who are members of application roles to claim and act upon tasks.

Application roles consist of users or other roles grouped logically for application-level authorizations. These roles are application-specific and are defined in the application Java policy store rather than the identity store. These roles are used by the application directly and are not necessarily known to a Java EE container.

Application roles define policy. Java permissions can be granted to application roles. Therefore, application roles define a set of permissions granted to them directly or indirectly through other roles (if a role is granted to a role). The policy can contain grants of application roles to enterprise groups or users. In the `jazn-data.xml` file of the file-based policy store, these roles are defined in `<app-role>` elements under `<policy-store>` and written to `system-jazn-data.xml` at the farm level during deployment. You can also define these roles after deployment using Oracle Enterprise Manager Fusion Middleware Control. You can set a task owner or approver to an application role at design time if the role has been previously deployed.

For more information about Oracle BPM Worklist, see [Section 26.2.1.6, "Task Forms."](#)

26.2.1.1.4 Ad Hoc Routing In processes dealing with significant variance, you cannot always determine all participants. Human workflow enables you to specify that a participant can invite other participants as part of performing the task.

For more information, see [Section 27.3.7.1.1, "Allowing All Participants to Invite Other Participants."](#)

26.2.1.1.5 Outcome-based Completion of Routing Flow By default, a task goes from starting to final participant according to the flow defined in the routing policy (as shown in [Figure 26-2](#)). However, sometimes a certain outcome at a particular step within a task's routing flow makes it unnecessary or undesirable to continue presenting the task to the next participants. For example, if an approval is rejected by the first manager, it does not need to be routed to the second manager. Human workflow supports specifying that a task or subtask be completed when a certain outcome occurs.

For more information, see [Section 27.3.7.1.2, "Stopping Routing of a Task to Further Participants."](#)

26.2.1.2 Static, Dynamic, and Rule-Based Task Assignment

There are different methods for assigning users, groups, and application roles to tasks.

- Assign tasks statically

You can assign users, groups, and application roles statically (or by browsing the identity service). The values can be either of the following:

- A single user, group, or application role (for example, `jstein`, `CentralLoanRegion`, or `ApproverRole`).
- A delimited string of users, groups, or application roles (for example, `jstein,wfaulk,cdickens`).

- Assign tasks dynamically

You can assign users, groups, and application roles dynamically using XPath expressions. These expressions enable you to dynamically determine the task participants at runtime. For example, you may have a business requirement to create a dynamic list of task approvers specified in a payload variable. The XPath expression can resolve to zero or more XML nodes. Each node value can be either of the following:

- A single user, group, or application role
- A delimited string of users, groups, or application roles. The default delimiter for the assignee delimited string is a comma (,).

For example, if the task has a payload message attribute named `po` within which the task approvers are stored, you can use the following XPath expression:

- `/task:task/task:payload/po:purchaseOrder/po:approvers`
- `ids:getManager('jstein', 'jazn.com')`

This returns the manager of `jstein`.

- `ids:getReportees('jstein', 2, 'jazn.com')`

This returns all reportees of `jstein` up to two levels.

- `ids:getUsersInGroup('LoanAgentGroup', false, 'jazn.com')`

This returns all direct and indirect users in the group `LoanAgentGroup`.

- Assign tasks with business rules

You can create the list of task participants with complex expressions. The result of using business rules is the same as using XPath expressions.

26.2.1.3 Task Stakeholders

A task has multiple stakeholders. Participants are the users defined in the assignment and routing section of the task definition. These users are the primary stakeholders that perform actions on the task.

In addition to the participants specified in the assignment and routing policy, human workflow supports additional stakeholders:

- Owner

This participant has business administration privileges on the task. This participant can be specified as part of the task definition or from the invoking process (and for a particular instance). The task owner can act upon tasks they own and also on behalf of any other participant. The task owner can change both the outcome of the task and the assignments.

For more information, see [Section 27.3.4.6, "Specifying a Task Owner"](#) to specify an owner in the Human Task Editor or [Section 27.4.4.2, "Specifying a Task Owner"](#) to specify an owner in the **Advanced** tab of the Human Task dialog.

- Initiator

The person who initiates the process (for example, the initiator files an expense report for approval). This person can review the status of the task using initiated task filters. Also, a useful concept is for including the initiator as a potential candidate for request-for-information from other participants.

For more information, see [Section 27.4.3.2, "Specifying the Task Initiator and Task Priority."](#)

- Reviewer

This participant can review the status of the task and add comments and attachments.

- Admin

This participant can view all tasks and take certain actions such as reassigning a test, suspending a task to handle errors, and so on. The task admin cannot change the outcome of a task.

While the task admin cannot perform the types of actions that a task participant can, such as approve, reject, and so on, this participant type is the most powerful because it can perform actions such as reassign, withdraw, and so on.

- Error Assignee

When an error occurs, the task is assigned to this participant (for example, the task is assigned to a nonexistent user). The error assignee can perform task recovery actions from Oracle BPM Worklist, the task form in which you perform task actions during runtime.

For more information, see [Section 27.3.7.4, "Configuring the Error Assignee."](#)

26.2.1.4 Task Deadlines

Human workflow supports the specification of deadlines associated with a task. You can associate the following actions with deadlines:

- Reminders:
The task can be reminded multiple times based on the time after the assignment or the time before the expiration.
- Escalation:
The task is escalated up the management hierarchy.
- Expiration:
The task has expired.
- Renewal:
The task is automatically renewed.

For more information, see [Section 27.3.9, "How to Escalate, Renew, or End the Task."](#)

26.2.1.5 Notifications

You can configure your human task to use notifications. Notifications enable you to alert interested users to changes in the state of a task during the task lifecycle. For example, a notification is sent to an assignee when a task has been approved or withdrawn.

You can specify for notifications to be sent to different types of participants for different actions. For example, you can specify the following:

- For the owner of a task to receive a notification message when a task is in error (for example, been sent to a nonexistent user).
- For a task assignee to receive a notification message when a task has been escalated.

You can specify the contents of the notification message and the notification channel to use for sending the message.

- Email
You can configure email notification messages to be actionable, meaning that a task assignee can act upon a task from within the email.
- Voice message
- Instant messaging (IM)
- Short message service (SMS)

For example, you may send the message shown in [Example 26–1](#) by email when a task assignee requests additional information before they can act upon a task:

Example 26–1 Email Message

```
For me to approve this task, more information is required to justify the need  
for this business trip
```

During runtime, you can mark a message sender's address as spam and also display a list of bad or invalid addresses. These addresses are automatically removed from the bad address list.

For more information about notifications, see the following:

- [Chapter 16, "Using the Notification Service"](#)
- [Section 27.3.10, "How to Specify Participant Notification Preferences"](#)

- [Part XI, "Using Oracle User Messaging Service"](#)

26.2.1.6 Task Forms

Task forms provide you with a way to interact with a task. Oracle BPM Worklist displays all worklist tasks that are assigned to task assignees in the task form. When you drill down into a specific task, the task form displays the contents of the task to the user's worklist. For example, an expense approval task may show a form with line items for various expenses, and a help desk task form may show details such as severity, problem location, and so on.

The integrated development environment of Oracle SOA Suite includes Oracle Application Development Framework (Oracle ADF) for this purpose. With Oracle ADF, you can design a task form that depicts the human task in the SOA composite application.

ADF-based task forms can be automatically generated. Advanced users can design their own task forms by using ADF data controls to lay out the content on the page and connect to the workflow service engine at execution time to retrieve task content and act on tasks.

You can create task forms in JSF, .NET, or any other client technologies using the APIs.

Integration with Microsoft Excel for initiating and acting on tasks is also provided.

For more information, see the following:

- [Chapter 28, "Designing Task Forms for Human Tasks."](#)
- [Chapter 29, "Using Oracle BPM Worklist"](#)

26.2.1.7 Advanced Concepts

This section describes advanced human workflow concepts.

26.2.1.7.1 Rule-based Routing You can use Oracle Business Rules to dynamically alter the routing flow. If used, each time a participant completes their step, the associated rules are invoked and the routing flow can be overridden from the rules.

For more information, see [Section 27.3.7.2, "Specifying Advanced Task Routing Using Business Rules."](#)

26.2.1.7.2 Rule-based Participant Assignment You can use Oracle Business Rules to dynamically build a list of users, groups, and roles to associate with a participant.

For more information, see [Section 27.3.6, "How to Assign Task Participants."](#)

26.2.1.7.3 Stages A stage is a way of organizing the approval process for blocks of participant types. You can have one or more stages in sequence or in parallel. Within each stage, you can have one or more participant type blocks in sequence or in parallel.

For more information, see [Section 27.3.6, "How to Assign Task Participants."](#)

26.2.1.7.4 Access Rules You can specify access rules that determine the parts of a task that assignees can view and update. For example, you can configure the task payload data to be read by assignees. This action enables only assignees (and nobody else) to have read permissions. No one, including assignees, has write permissions.

For more information, see [Section 27.3.11.1, "Specifying Access Policies on Task Content."](#)

26.2.1.7.5 Callbacks While human workflow supports detailed behavior that can be declaratively specified, in some advanced situations, more extensible behavior may be required. Task callbacks enable such extensibility; these callbacks can either be handled in the invoking BPEL process or a Java class.

For more information, see [Section 27.3.14.1, "Specifying Callback Classes on Task Status."](#)

26.2.1.8 Reports and Audit Trails

Oracle BPM Worklist provides several out-of-the-box reports for task analysis:

- **Unattended tasks**
Analysis of tasks assigned to users' groups or reportees' groups that have not yet been acquired.
- **Tasks priority**
Analysis of tasks assigned to a user, reportees, or their groups, based on priority.
- **Tasks cycle time**
Analysis of the time taken to complete tasks from assignment to completion based on users' groups or reportees' groups.
- **Tasks productivity**
Analysis of assigned tasks and completed tasks in a given time period for a user, reportees, or their groups.
- **Tasks time distribution**
The time an assignee takes to perform a task.

You can view an audit trail of actions performed by the participants in the task and a snapshot of the task payload and attachments at various points in the workflow. The short history for a task lists all versions created by the following tasks:

- Initiate task
- Reinitiate task
- Update outcome of task
- Completion of task
- Erring of task
- Expiration of task
- Withdrawal of task
- Alerting of task to the error assignee

For more information, see [Chapter 29, "Using Oracle BPM Worklist."](#)

26.2.2 Introduction to the Stages of Human Workflow Design

Human workflow modeling consists of three stages of modeling, as described in [Table 26–1](#).

Table 26–1 Stages of Human Workflow Modeling

Step	Description	For More Information...
1	You create and define contents of the human task in the Human Task Editor, including defining a participant type, routing policy, escalation and expiration policy, notification, and so on.	Section 27.2.1, "Create a Human Task Definition."
2	You associate the human task definition with a BPEL process. The BPEL process integrates a series of activities (including the human task activity) and services into an end-to-end process flow.	Section 27.2.2, "Associate the Human Task Definition with a BPEL Process."
3	You create a task form. This form displays the task details on which you act at runtime in Oracle BPM Worklist.	Section 27.2.3, "Generate the Task Form."

26.3 Introduction to Human Workflow Features

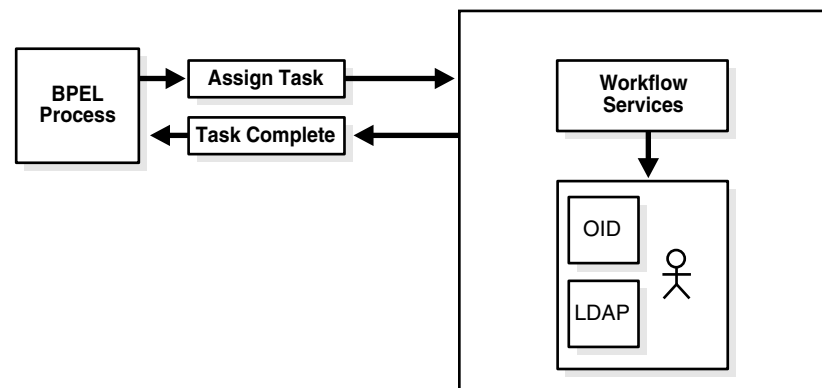
This section provides an introduction to use cases for human workflow. After that, a tutorial guides you through the design of a human task from start to finish.

26.3.1 Human Workflow Use Cases

The following sections describe multiple use cases for workflow services.

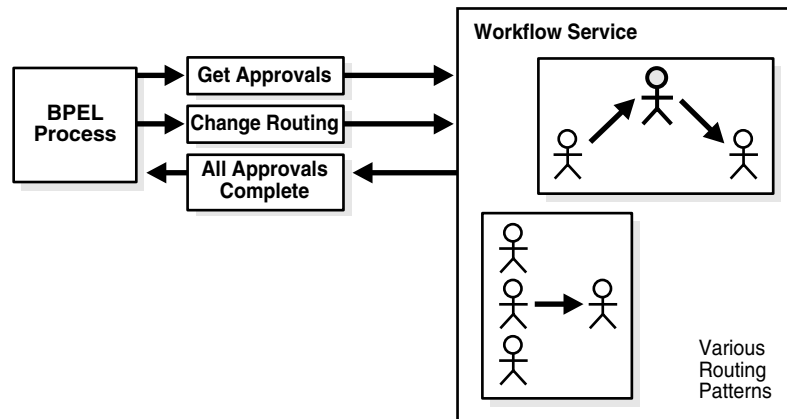
26.3.1.1 Task Assignment to a User or Role

A vacation request process may start with getting the vacation details from a user and then routing the request to their manager for approval. User details and the organizational hierarchy can be looked up from a user directory or identity store. This scenario is shown in [Figure 26–3](#).

Figure 26–3 Assigning Tasks to a User or Role from a Directory

26.3.1.2 Use of the Various Participant Types

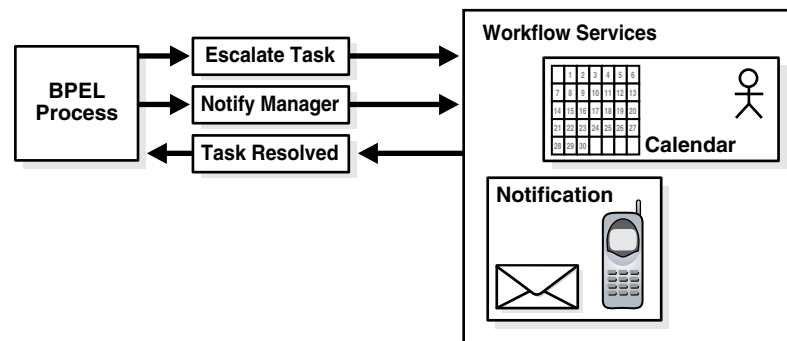
A task can be routed through multiple users with a group vote, management chain, or sequential list of approvers participant type. For example, consider a loan request that is part of the loan approval flow. The loan request may first be assigned to a loan agent role. After a specific loan agent acquires and accepts the loan, the loan may be routed further through multiple levels of management if the loan amount is greater than \$100,000. This scenario is shown in [Figure 26–4](#).

Figure 26–4 Flow Patterns and Routing Policies

You can use these types as building blocks to create complex workflows.

26.3.1.3 Escalation, Expiration, and Delegation

A high-priority task can be assigned to a certain user or role based on the task type through use of custom escalation functions. However, if the user does not act on it in a certain time, the task may expire and in turn be escalated to the manager for further action. As part of the escalation, you may also notify the users by email, telephone voice message, or SMS. Similarly, a manager may delegate tasks from one reportee to another to balance the load between various task assignees. All tasks defined in BPEL have an associated expiration date. Additionally, you may specify escalation or renewal policies, as shown in [Figure 26–5](#). For example, consider a support call, which is part of a help desk service request process. A high-priority task may be assigned to a certain user, and if the user does not respond in two days, the task is routed to the manager for further action.

Figure 26–5 Escalation and Notification

26.3.1.4 Automatic Assignment and Delegation

A user may decide to have another user perform tasks on their behalf. Tasks can be explicitly delegated from the Oracle BPM Worklist or can be automatically delegated. For example, a manager sets up a vacation rule saying that all their high priority tasks are automatically routed to one of their direct reports while the manager is on vacation. In some cases, tasks can be routed to different individuals based on the content of the task. Another example of automatic routing is to allocate tasks among multiple individuals belonging to a group. For example, a help desk supervisor decides to allocate all tasks for the western region based on a round robin basis or

assign tasks to the individual with the lowest number of outstanding tasks (the least busy).

26.3.1.5 Dynamic Assignment of Users Based on Task Content

An employee named James in the human resources department requests new hardware that costs \$5000. The company may have a policy that all hardware expenses greater than \$3000 must go through manager and vice president approval, and then review by the director of IT. In this scenario, the workflow can be configured to automatically determine the manager of James, the vice president of the human resources department, and the director of IT. The purchase order is routed through these three individuals for approval before the hardware is purchased.

26.3.2 Designing a Human Task from Start to Finish

This section guides you through design of your first human task.

This sample describes how an employee submits a vacation request that is automatically routed to their manager for approval. Once the manager responds (approved or rejected), a notification is sent to the employee.

This sample illustrates creation of a SOA composite application with two components:

- A BPEL process
- A human task, for approving a vacation request submitted by an employee

This example highlights the use of the following:

- Using the SOA Composite Editor and Human Task Editor
- Modeling a single approval workflow using Oracle BPEL Designer
- Creating an Oracle ADF-based Oracle BPM Worklist
- Using Oracle BPM Worklist to view and respond to the task

26.3.2.1 Prerequisites

This tutorial makes the following assumptions:

- Oracle SOA Suite is installed on a host on which the SOA Infrastructure is configured.
 - You are familiar with basic BPEL constructs, including BPEL activities and partner links, and basic XPath functions. Familiarity with the SOA Composite Editor and Oracle BPEL Designer, the environment for designing and deploying BPEL processes, is also assumed.
1. Create a file named `VacationRequest.xsd` with the following syntax. This file includes the schema for the vacation request and subsequent response.

```
<schema attributeFormDefault="qualified" elementFormDefault="qualified"
  targetNamespace="http://xmlns.oracle.com/VacationRequest"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="VacationRequestProcessRequest">
    <complexType>
      <sequence>
        <element name="creator" type="string"/>
        <element name="fromDate" type="date"/>
        <element name="toDate" type="date"/>
        <element name="reason" type="string"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

```
</element>
<element name="VacationRequestProcessResponse">
  <complexType>
    <sequence>
      <element name="result" type="string"/>
    </sequence>
  </complexType>
</element>
</schema>
```

Note: The `VacationRequest.xsd` file is also available for download as part of tutorial workflow-100-VacationRequest. See [Section 26.3.3, "Additional Tutorials"](#) for information on downloading this and other tutorials.

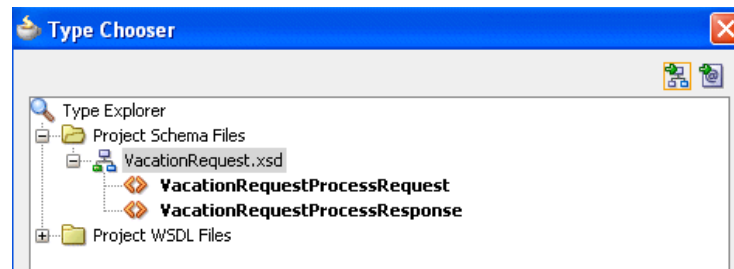
26.3.2.2 How to Create the Vacation Request Process

In this tutorial, you create a new application and SOA project and design the human task to send a vacation request to a manager for approval or rejection. You also create a second application and project in which you create an Oracle ADF-based task form from which to act upon the vacation request.

26.3.2.2.1 Creating an Application and a Project with a BPEL Process

To create an application and a project with a BPEL process:

1. Start Oracle JDeveloper.
2. From the **File** main menu, select **New > Applications > SOA Application**.
3. Click **OK**.
4. In the **Application Name** field, enter `VacationRequest`, and click **Next**.
5. In the **Project Name** field, enter `VacationRequest`, and click **Next**.
6. In the **Composite Template** list, select **Composite with BPEL Process**, and click **Finish**.
7. The Create BPEL Process dialog appears.
8. In the **Name** field, enter `VacationRequestProcess`.
9. Go to the bottom of the Create BPEL Process dialog.
10. To the right of the **Input** field, click the **Search** icon.
The Type Chooser dialog appears.
11. In the upper right corner, click the **Import Schema File** icon.
The Import Schema File dialog appears.
12. Browse for and select the **VacationRequest.xsd** file you created in [Section 26.3.2.1, "Prerequisites."](#)
13. Click **OK** until you are returned to the Type Chooser dialog, as shown in [Figure 26–6](#).

Figure 26–6 Type Chooser Dialog with the Request and Response Elements

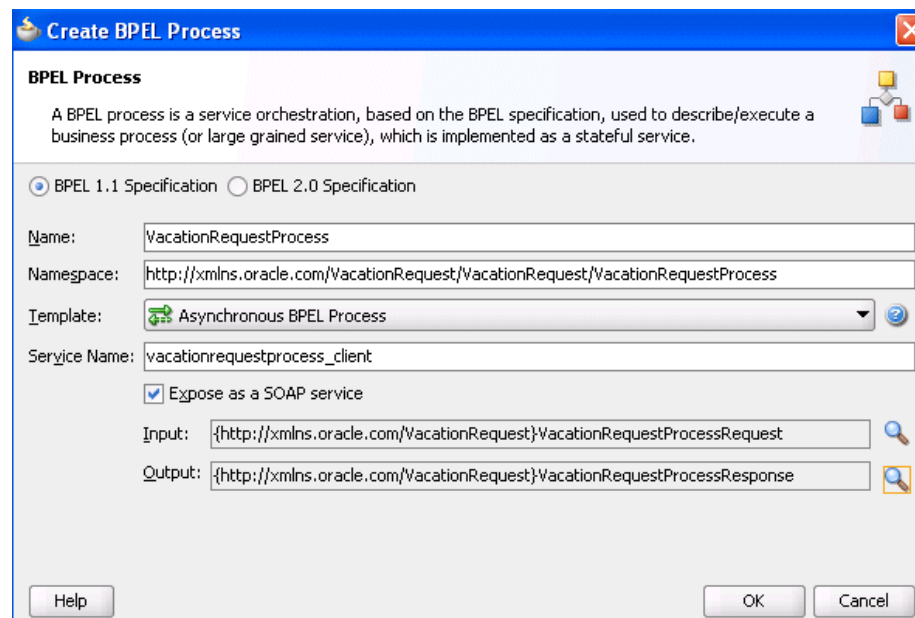
14. Select the input element **VacationRequestProcessRequest**, and click **OK**.

You are returned to the Create BPEL Process dialog.

15. To the right of the **Output** field, click the **Search** icon.

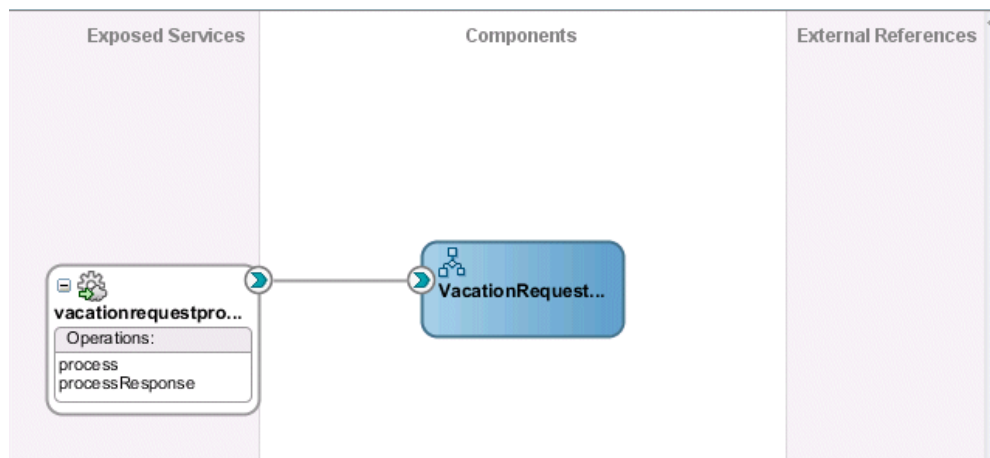
16. Select the output element **VacationRequestProcessResponse**, and click **OK**.

You are returned to the Create BPEL Process dialog, as shown in [Figure 26–7](#).

Figure 26–7 BPEL Process Dialog

17. Accept the default values for all other settings, and click **OK**.

A BPEL process service component is created in the SOA Composite Editor, as shown in [Figure 26–8](#). Because **Expose as a SOAP service** was selected in the Create BPEL Process dialog, the BPEL process is automatically connected with a service binding component. The service exposes the SOA composite application to external customers.

Figure 26–8 BPEL Process in SOA Composite Editor

For more information about service components and the SOA Composite Editor, see [Chapter 2, "Developing SOA Composite Applications with Oracle SOA Suite."](#)

26.3.2.2.2 Create the Human Task Service Component

You are now ready to create the human task service component in which you design your human task.

To create the human task service component:

1. From the **Service Components** section of the Component Palette, drag a **Human Task** into the SOA Composite Editor.

The Create Human Task dialog appears.

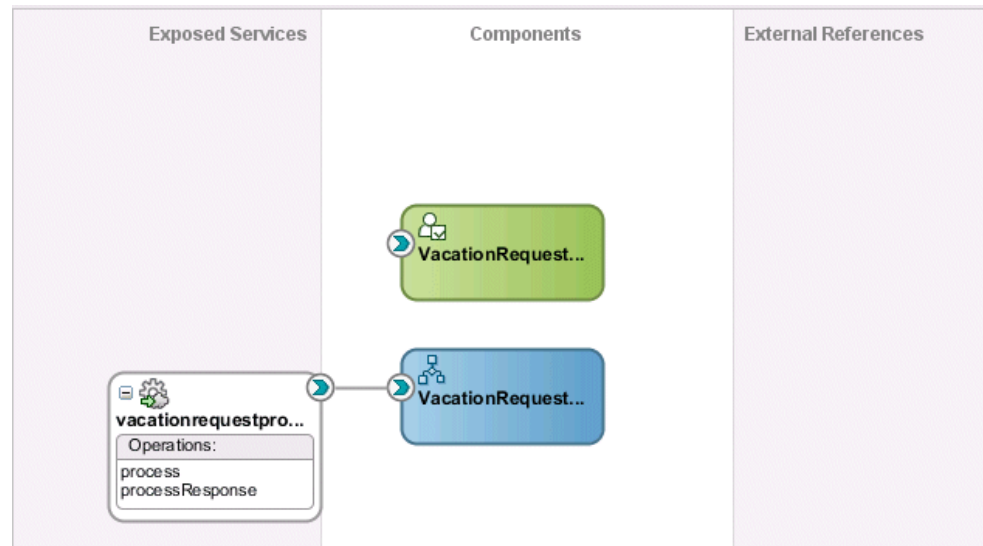
2. Enter the details described in [Table 26–2](#).

Table 26–2 Create Human Task Dialog Fields and Values

Field	Value
Name	Enter VacationRequestTask.
Namespace	Accept the default value.
Create Composite Service with SOAP Bindings	Do <i>not</i> select the checkbox. Instead, you create a human task that you later associate with the BPEL process you created in Section 26.3.2.2.1, "Creating an Application and a Project with a BPEL Process." The BPEL process was created with an automatically-bound web service.

3. Click **OK**.

The **Human Task** icon appears in the SOA Composite Editor above the BPEL process, as shown in [Figure 26–9](#).

Figure 26–9 Human Task Icon in SOA Composite Editor

4. Double-click the **Human Task** icon.

The Human Task Editor appears. You are now ready to begin design of your human task.

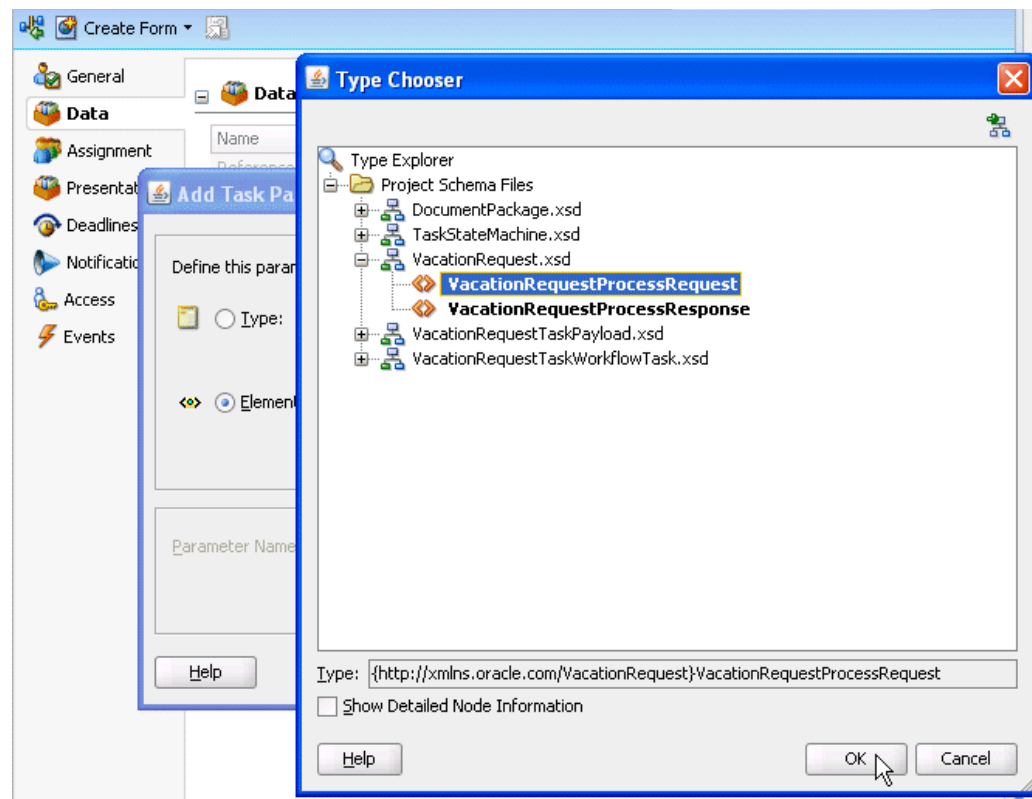
26.3.2.2.3 Designing the Human Task

To design the human task:

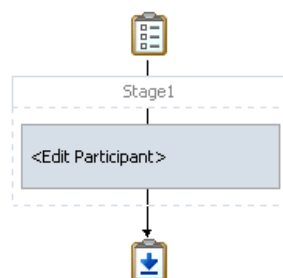
1. In the **Task Title** field, enter Request for Vacation.
2. Accept the default values for outcomes (**APPROVE** and **REJECT**). For this task, these outcomes represent the two choices the manager has for acting on the vacation request.
3. Click the **Data** tab on the left side of the editor.
4. Click the **Add** icon to specify the task payload.
5. Select **Add string parameter**.

The Add Task Parameter dialog is displayed. You now create parameters to represent the elements in your XSD file. This makes the payload data available to the workflow task.

6. Select **Element**.
7. To the right of the **Element** field, click the **Search** icon.
The Type Chooser dialog appears.
8. Expand and select **Project Schema Files > VacationRequest.xsd > VacationRequestProcessRequest**, and click **OK**. [Figure 26–10](#) provides details.

Figure 26–10 Type Chooser Dialog

9. Ensure that the **Editable via worklist** checkbox is selected. This provides you with the option to modify this parameter during runtime from Oracle BPM Worklist.
10. Click **OK** on the Add Task Parameter dialog.
11. Click the **Assignment** tab on the left side of the editor.
12. Highlight the **<Edit participant>** box below **Stage1**, as shown in [Figure 26–11](#).

Figure 26–11 Assignment and Routing Policy

13. At the top of the Human Task Editor, click the **Edit** icon.

The Edit Participant Type dialog appears. You now add participants to this task. As described in [Section 26.2.1.1.2, "Participant Type,"](#) Oracle SOA Suite provides several out-of-the-box patterns known as participant types for addressing specific business needs.

14. Accept the default participant type of **Single** that displays in the **Type** list. You select this type because a single assignee, the manager, acts on the vacation request task.

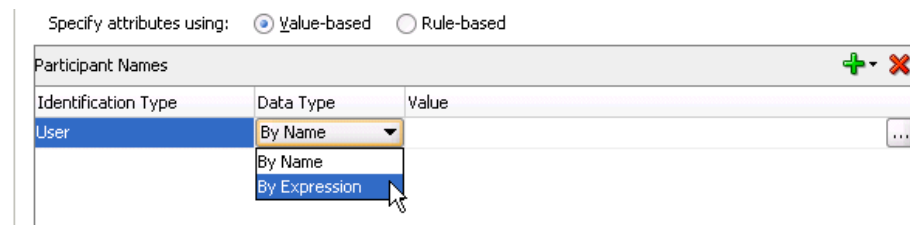
15. In the **Participant Names** table, click the **Add** icon, and select **Add User**.

This participant type acts alone on the task.

16. Click the **Data Type** column, and select **By Expression** from the list that is displayed. [Figure 26–12](#) provides details.

This action enables the task to be assigned dynamically by the contents of the task. The employee filing the vacation request comes from the parameter passed to the task (the creator element in the XSD file you imported in [Section 26.3.2.2.1, "Creating an Application and a Project with a BPEL Process"](#)). The task is automatically routed to the employee's manager.

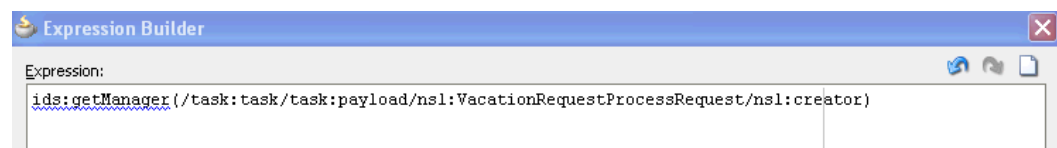
Figure 26–12 Selection of By Expression from the Data Type Column



17. In the **Value** column, click the **Browse** icon (the dots) to invoke the Expression Builder dialog.
18. In the dropdown list in the **Functions** section, select **Identity Service Functions**.
19. Select **getManager**. This function gets the manager of the user who created the vacation request task.
20. Above the **Functions** section, click **Insert into Expression**.
21. Place the cursor between the parentheses of the function.
22. In the **Schema** section, expand **task:task > task:payload > ns1:VacationRequestProcessRequest > ns1:creator**.
where **ns1** is the namespace for this example; your namespace may be different.
23. Click **Insert into Expression**.

The Expression Builder dialog displays the XPath expression in the **Expression** section. [Figure 26–13](#) provides details.

Figure 26–13 XPath Expression



24. Click **OK** to exit the Expression Builder dialog.
25. Click **OK** to exit the Add Participant Type dialog.
26. From the **File** menu, select **Save All**.

26.3.2.2.4 Associating the Human Task and BPEL Process Service Components

You are now ready to associate your human task with the BPEL process you created in [Section 26.3.2.2.1, "Creating an Application and a Project with a BPEL Process."](#)

To associate the human task and BPEL process service component:

1. In the Application Navigator, double-click **composite.xml**.
2. Double-click the **VacationRequestProcess** BPEL process service component in the SOA Composite Editor.

The BPEL process displays in Oracle BPEL Designer.

3. In the Component Palette, expand **SOA Components**.
4. Drag a **Human Task** beneath the **receiveInput** receive activity.
5. Double-click the activity.

The Human Task dialog appears.

6. From the **Task Definition** list, select the **VacationRequestTask** task you created (if it is not currently displaying).

The dialog refreshes as shown in [Figure 26–14](#) to display additional fields.

Figure 26–14 Human Task Dialog

Task Parameters	BPEL Variable
VacationRequestProcessRequest	...

7. In the **BPEL Variable** column, click the **Browse** icon (dots) shown in [Figure 26–15](#).

Figure 26–15 BPEL Variable Entry

The **Human Task** dialog box is shown with the **General** tab selected. The **Task Definition** is set to **VacationRequestTask**. The **Task Title** field contains the text: "Enter text to override default value from .task". Below this, there is a hint: "E.g., Vacation Request for <%bpws:getVariableData(...)%">". The **Initiator** field is empty. The **Priority** is set to **3**. The **Task Parameters** table is shown with the following data:

Task Parameters	BPEL Variable
VacationRequestProcessRequest	

A red box highlights the empty **BPEL Variable** field for the **VacationRequestProcessRequest** parameter.

The Task Parameters dialog appears.

8. From the **Type** list, select **Variable**.
9. Expand **Process > Variables > inputVariable > payload > ns1:VacationRequestProcessRequest**. Figure 26–16 provides details.

Figure 26–16 Variable Selection

The **Task Parameters** dialog box is shown with the **From** tab selected. The **Type** is set to **Variable**. The variable selection tree is expanded to show the following structure:

- Variables
 - Process
 - Variables
 - inputVariable
 - payload
 - ns1:VacationRequestProcessRequest : <anonymous> (selected)
 - ns1:creator : string
 - ns1:fromDate : date
 - ns1:toDate : date
 - ns1:reason : string

The **outputVariable** and **Scope - HumanTask1** are also visible in the tree. The **Show Detailed Node Information** checkbox is unchecked. The **XPath** field contains the text: `/ns1:VacationRequestProcessRequest`. The **Help**, **OK**, and **Cancel** buttons are at the bottom.

10. Click **OK**.

When complete, the dialog looks as shown in [Figure 26–17](#):

Figure 26–17 BPEL Variable

The image shows a 'Human Task' dialog box with three tabs: 'General', 'Advanced', and 'Annotations'. The 'General' tab is selected. It contains the following fields:

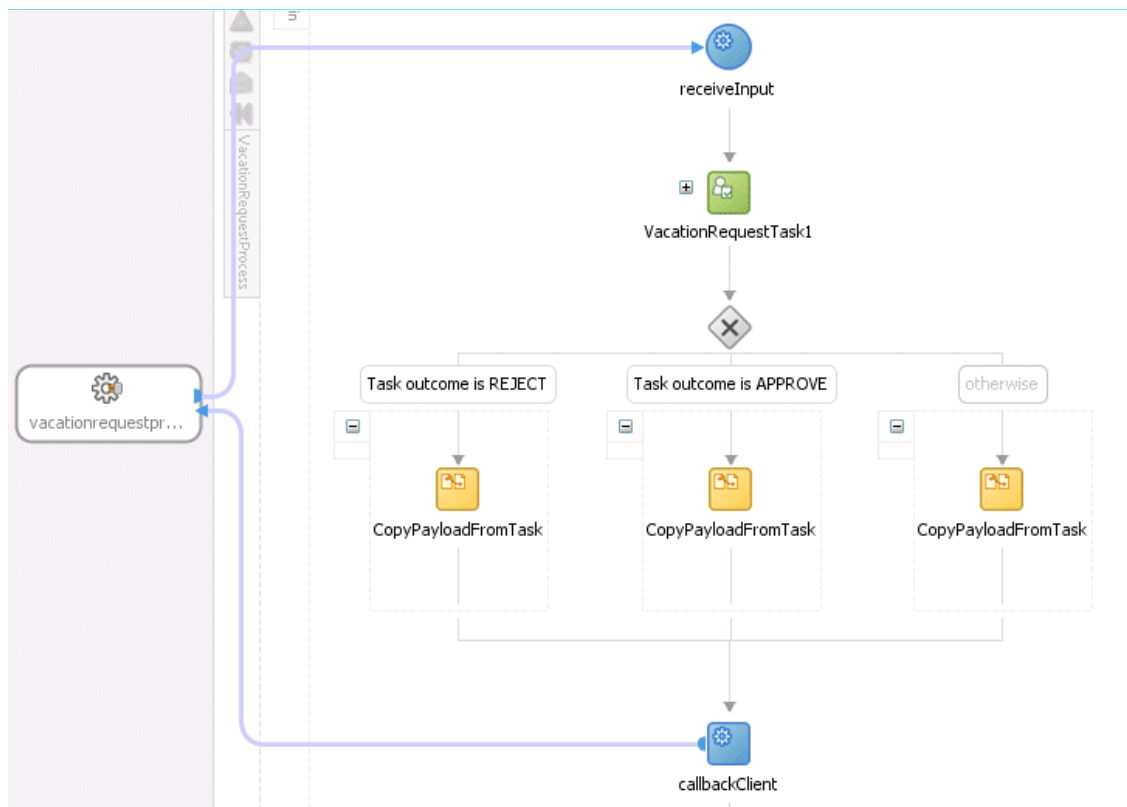
- Task Definition:** VacationRequestTask
- Task Title:** VacationRequestTask (with a hint: E.g., Vacation Request for <%bpws:getVariableData(...)%)>)
- Initiator:** (empty field)
- Priority:** 3 (dropdown menu)

Below these fields is a table with two columns: 'Task Parameters' and 'BPEL Variable'.

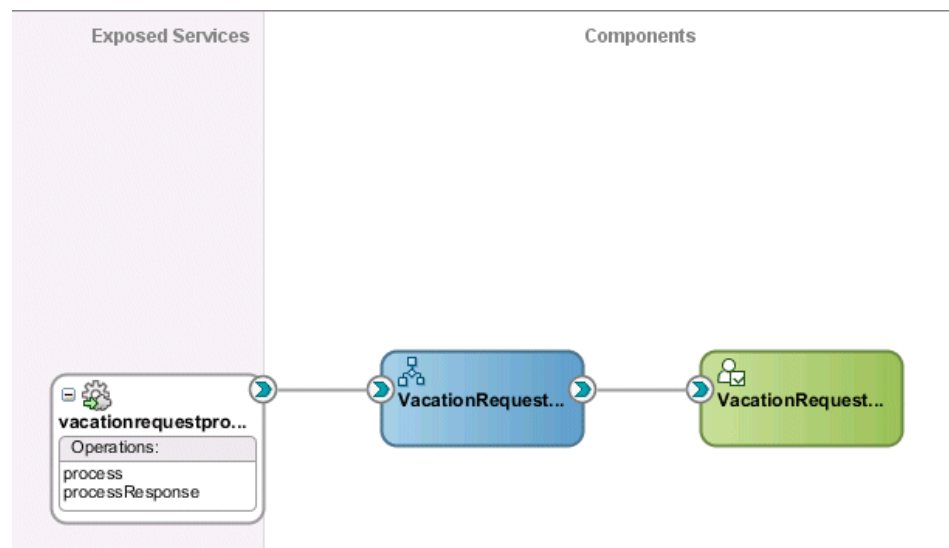
Task Parameters	BPEL Variable
VacationRequestProcessRequest	/ns1:VacationRequestProcessRequest

11. Click **OK** to close the Human Task dialog.

The human task activity appears as shown in [Figure 26–18](#).

Figure 26–18 Human Task and Partner Links in Oracle BPEL Designer

12. Return to the SOA Composite Editor and note that the BPEL process and human task service components have been automatically connected. [Figure 26–19](#) provides details.

Figure 26–19 SOA Composite Editor

13. From the **File** menu, select **Save All**.

26.3.2.2.5 Creating an Application Server Connection

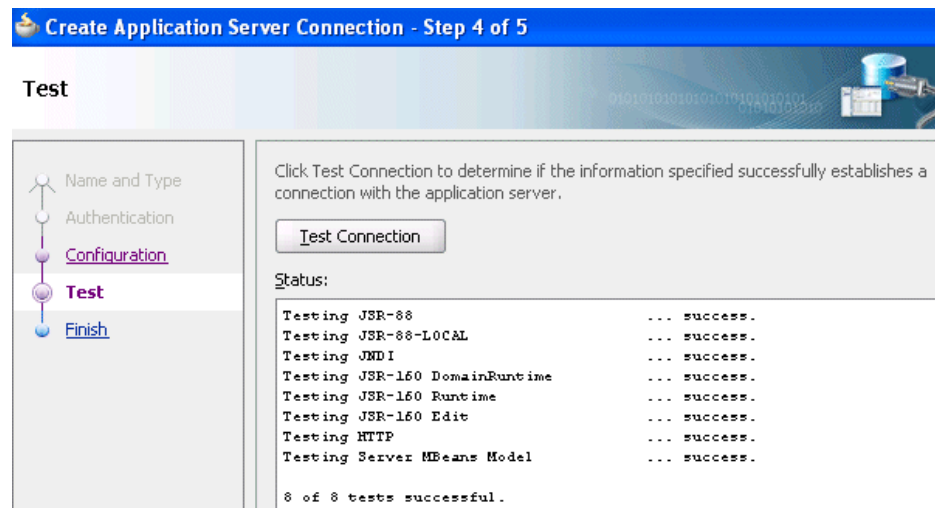
You are now ready to create a connection to the application server on which Oracle SOA Suite is installed and configured with the SOA Infrastructure. These instructions describe how to create a connection to Oracle WebLogic Server. For information about creating a connection to other application servers such as IBM WebSphere Server, see *Oracle Fusion Middleware Third-Party Application Server Guide*.

To create an application server connection

1. From the **File** main menu, select **New > Connections > Application Server Connection**.
2. Click **OK**.
3. In the **Connection Name** field, enter a connection name.
4. From the **Connection Type** list, select **WebLogic 10.3**.
5. Click **Next**.
6. In the **Username** field, enter `weblogic`.
7. In the **Password** field, enter the password for connecting to the application server.
8. Click **Next**.
9. Enter the hostname for the application server that is configured with the SOA Infrastructure.
10. In the **Weblogic Domain** field, enter the Oracle WebLogic Server domain.
11. Click **Next**.
12. Click **Test Connection**.

If successful, the message shown in [Figure 26–20](#) is displayed.

Figure 26–20 Connection Success



13. Click **Finish**.
14. From the **File** menu, select **Save All**.

26.3.2.2.6 Deploying the SOA Composite Application

You are now ready to deploy to the application server on which you created the connection.

To deploy the SOA composite application

1. In the Application Navigator, right-click the **VacationRequest** project and select **Deploy > VacationRequest**.
2. Follow the pages of the deployment wizard to deploy the project.

The project is deployed.

For more information about deployment, see [Section 40.7, "Deploying SOA Composite Applications."](#)

26.3.2.2.7 Initiating the Process Instance**To initiate the process instance:**

1. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for instructions on accessing the Test Web Service page for initiating the process instance.

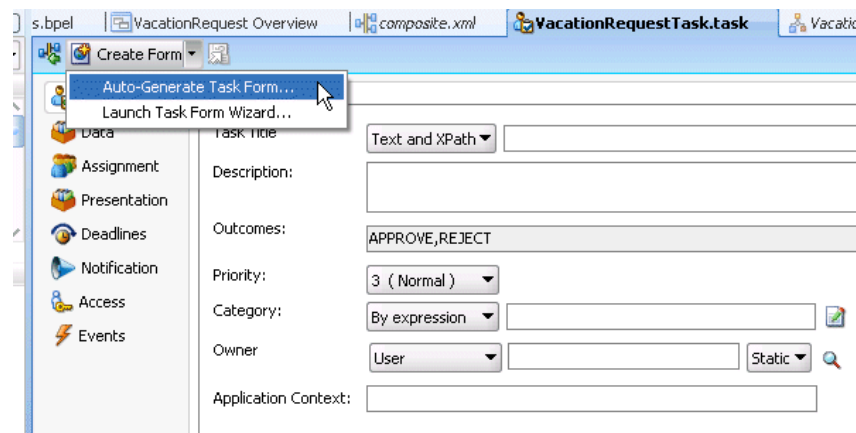
26.3.2.2.8 Creating a Task Form Project

You are now ready to create a project for the task form. This is a separate project from the one in which you created the human task.

To create a task form project:

1. Double-click the **VacationRequestTask** human task.
The Human Task Editor is displayed.
2. From the **Create Form** menu at the top, select **Auto-Generate Task Form**.
[Figure 26–21](#) provides details.

Figure 26–21 Task Form Creation



The Create Project dialog appears.

3. In the **Project Name** field, enter `VacationRequestTaskFlow`, and click **OK**.
4. From the **File** main menu, select **Save All**.

26.3.2.2.9 Acting on the Task in Oracle BPM Worklist**To resolve the task in Oracle BPM Worklist:**

1. Go to Oracle BPM Worklist:

`http://hostname:7001/integration/worklistapp`

2. Log in to Oracle BPM Worklist.
3. Resolve the task.

26.3.2.2.10 Deploying the Task Form

To deploy the task form:

1. In the Application Navigator, right-click the **VacationRequestTaskFlow** project and select **Deploy > VacationRequestTaskFlow**.

2. Follow the pages of the deployment wizard to deploy the task form.

The task form is deployed.

For more information about deployment, see [Section 40.7, "Deploying SOA Composite Applications."](#)

3. Return to Oracle BPM Worklist.
4. Note that the task form now appears at the bottom of Oracle BPM Worklist.

26.3.3 Additional Tutorials

In addition to the vacation request use case, other tutorials are available at the following URL:

<https://soasamples.samplecode.oracle.com>

[Table 26–3](#) provides an overview of some samples. All samples show the use of worklist applications and workflow notifications. For the complete list of samples, visit the URL.

Table 26–3 End-to-End Examples

Sample	Description	Name
Demo Community Seed Application	Performs demo community seeding. This is a prerequisite for all other workflow samples.	workflow-001-DemoCommunitySeedApp
Vacation Request	Provides a sample in which a user submits a vacation request that gets assigned to their manager for approval or rejection. This sample also describes how to create Oracle ADF task forms for the vacation request to act on the task.	workflow-100-VacationRequest
Sales Quote Request	Provides a complex workflow sample with chaining of multiple tasks.	workflow-102-SalesQuote
Contract Approval	Provides a sample of approving a contract. This sample uses digital signatures for tasks.	workflow-104-ContractApproval
Iterative Design	Provides a sample in which a workflow task can be passed multiple times between assignees during the design process. Advanced routing rules implement the routing behavior.	workflow-106-IterativeDesign

Table 26–3 (Cont.) End-to-End Examples

Sample	Description	Name
Workflow Customizations	Demonstrates how to deploy customizations to workflow service APIs, such as custom resource strings for task attributes, view names, and so on.	workflow-110-workflowCustomizations
MLS Sample	Demonstrates the setting up of a task with multiple translations for the task title.	workflow-114-MLSSample
Workflow Event Callback	Demonstrates the use of the workflow event callback. Workflow events generated by task lifecycle events are consumed by an Oracle Mediator.	workflow-116-WorkflowEventCallback
User Config Data Migrator	Moves user configurations (views, mapped attributes, and so on) from one instance to another through an intermediate export file.	workflow-117-UserConfigDataMigrator
Java Samples	Provides an assortment of samples that use Java to interact with human workflow.	workflow-118-JavaSamples

26.4 Introduction to Human Workflow Architecture

This section provides an overview of human workflow architecture. The following topics are discussed:

- The services that perform a variety of operations in the lifecycle of a task, such as querying tasks for a user, retrieving metadata information related to a task, and so on.
- The two ways to use a human task:
 - Associated with a BPEL process service component
 - Used in standalone mode
- The role of the service engine in the life of a human task

26.4.1 Human Workflow Services

Starting with release 11g, all human task metadata is stored and managed in the Metadata Service (MDS) repository. The workflow service consists of many services that handle various aspects of human interaction with a business process.

[Figure 26–22](#) shows the following workflow service components:

- Task Service:

The task service provides task state management and persistence of tasks. In addition to these services, the task service exposes operations to update a task, complete a task, escalate and reassign tasks, and so on. The task service is used by 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 through any of the following channels: email, telephone voice message, IM, and SMS. See [Section 31.2, "Notifications from Human Workflow"](#) for more information.

- User Metadata Service

The user metadata service manages metadata related to workflow users, such as user work queues, preferences, vacations, and delegation rules.

- Runtime Config Service

The runtime config service provides methods for managing metadata used in the task service runtime environment. It principally supports management of task payload mapped attribute mappings.

- Evidence service

The evidence service supports storage and nonrepudiation of digitally-signed workflow tasks.

Figure 26–22 Workflow Services Components

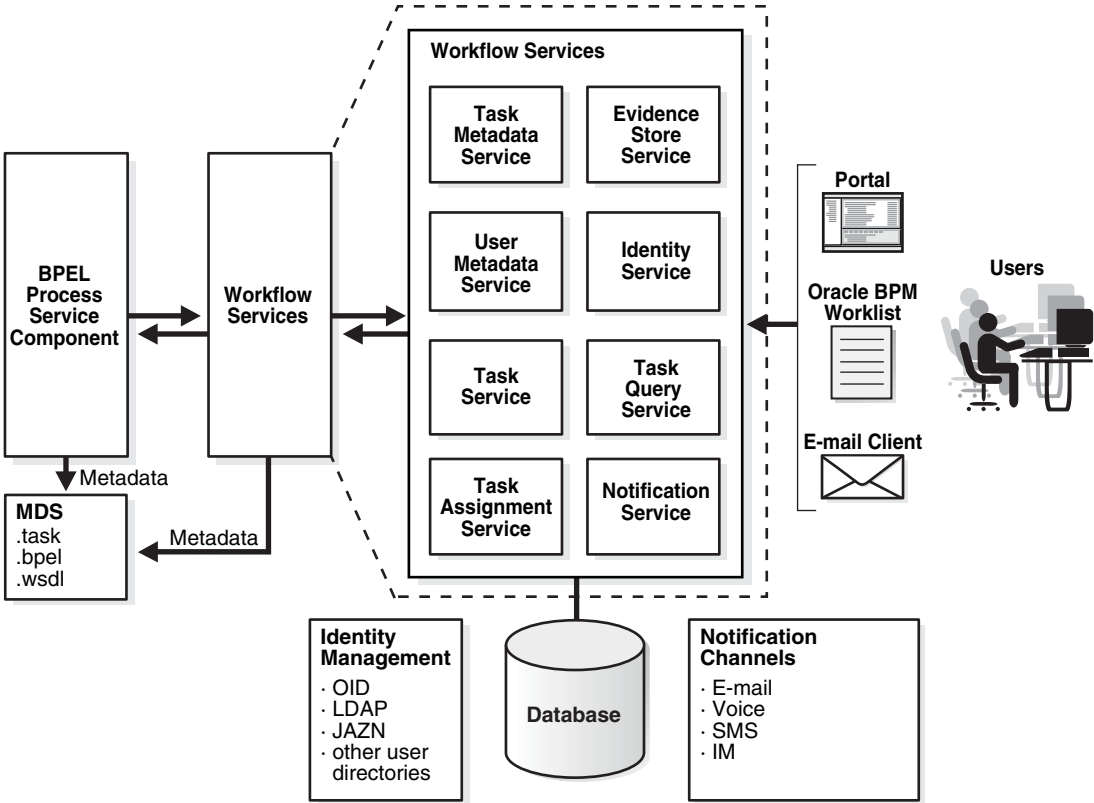
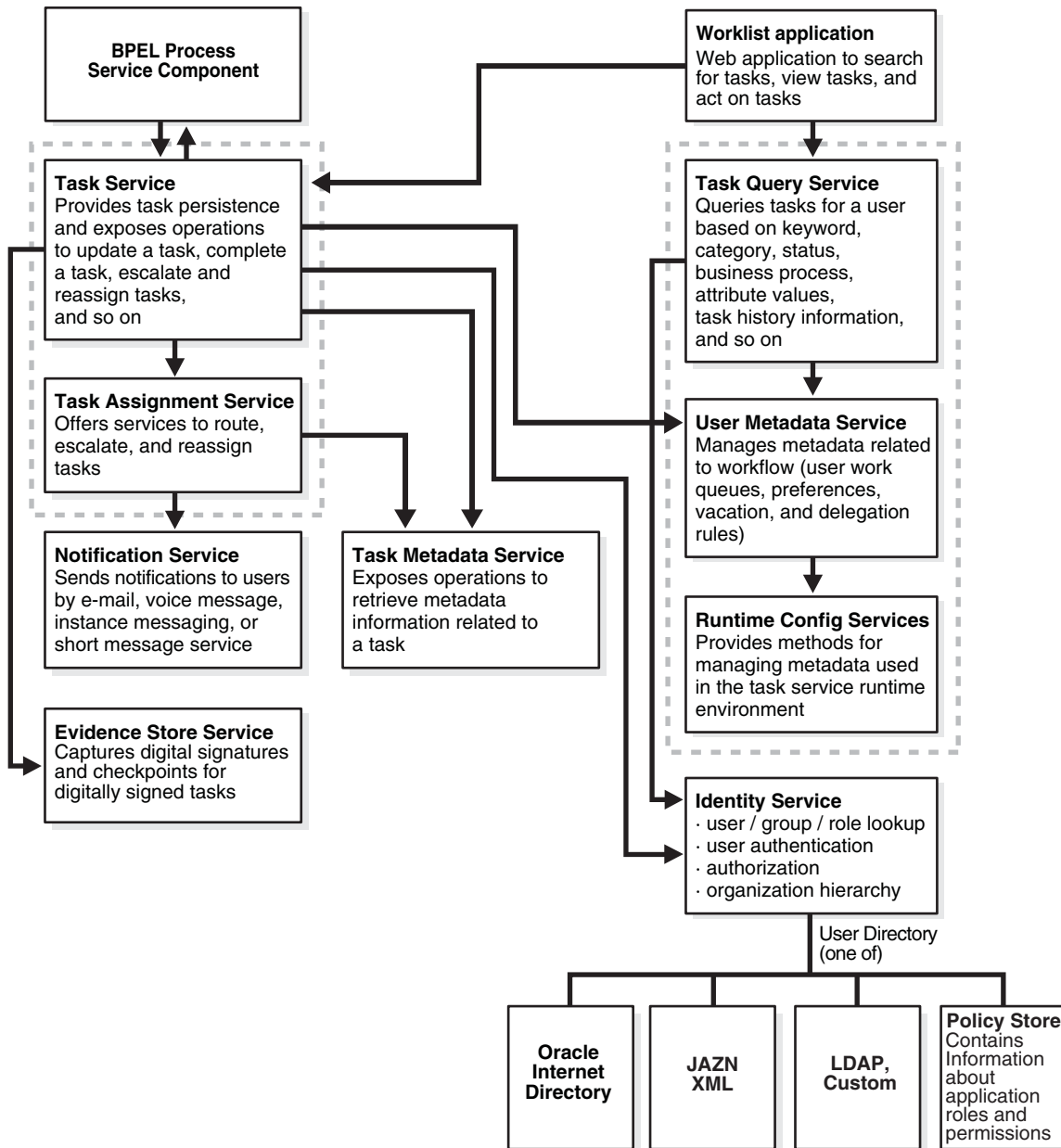


Figure 26–23 shows the interactions between the services and the business process.

Figure 26–23 Workflow Services and Business Process Interactions

26.4.2 Use of Human Task

There are two ways in which to use a human task:

- Human task associated with a BPEL process

In most cases, you associate your human task with a BPEL process. The BPEL process integrates a series of activities (including the human task activity) and services into an end-to-end process flow.

- Standalone human task

You can also create the human task as a standalone component only in the SOA Composite Editor and not associate it with a BPEL process. Standalone human task service components are useful for environments in which there is no need for

any automated activity in an application. In the standalone case, the client can create the task themselves.

26.4.3 Service Engines

During runtime, the business logic and processing rules of the human task service component are executed by the human workflow service engine. Each service component (BPEL process, human workflow, decision service (business rules), and Oracle Mediator) has its own service engine container for performing these tasks. All human task service components, regardless of the SOA composite application of which they are a part, are executed in this single human task service engine.

For more information about configuring, monitoring, and managing the human workflow service engine, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

Designing Human Tasks

This chapter describes how to design human tasks. It introduces the Human Task Editor to use for modeling task metadata, routing and assignment policies, escalation policies, expiration policies, and notification settings.

This chapter includes the following sections:

- [Section 27.1, "Introduction to Human Task Design Concepts"](#)
- [Section 27.2, "Introduction to the Modeling Process"](#)
- [Section 27.3, "Creating the Human Task Definition with the Human Task Editor"](#)
- [Section 27.4, "Associating the Human Task Service Component with a BPEL Process"](#)

For information about troubleshooting human workflow issues, see section "Human Workflow Troubleshooting" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

27.1 Introduction to Human Task Design Concepts

To use the Human Task Editor, you must understand human task design concepts, including the following:

- The types of users to which to assign tasks
- The methods by which to assign users to tasks (statically, dynamically, or rule-based)
- The task participant types available for modeling a task to which you assign users
- The options for creating lists of task participants
- The participants involved in the entire life cycle of a task

For information about human task concepts, see [Chapter 26, "Getting Started with Human Workflow."](#)

27.2 Introduction to the Modeling Process

Oracle SOA Suite provides a graphical tool, known as the Human Task Editor, for modeling your task metadata. The modeling process consists of the following:

- Creating and modeling a human task service component in the SOA Composite Editor
- Associating it with a BPEL process

- Generating the task form for displaying the human task during runtime in Oracle BPM Worklist.

This section provides a brief overview of these modeling tasks and provides references to specific modeling instructions.

For more information about using the SOA Composite Editor, see [Chapter 2, "Developing SOA Composite Applications with Oracle SOA Suite."](#)

For information about available samples, see [Section 26.3.2, "Designing a Human Task from Start to Finish."](#)

27.2.1 Create a Human Task Definition

You define the metadata for the human task in either of two ways:

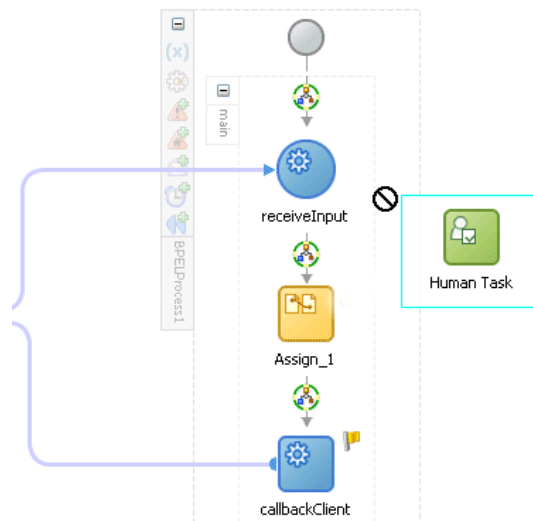
- By dragging a human task from the Component Palette into a BPEL process in Oracle BPEL Designer and clicking the **Add** icon in the Create Human Task dialog that automatically is displayed. This displays a dialog for creating the human task service component. When creation is complete, the Human Task Editor is displayed.
- By dragging a human task service component from the Component Palette into the SOA Composite Editor. This displays a dialog for creating the human task component. When creation is complete, the Human Task Editor is displayed.

The Human Task Editor enables you to specify human task metadata, such as task outcome, payload structure, assignment and routing policy, expiration and escalation policy, notification settings, and so on. This information is saved to a metadata task configuration file with a `.task` extension. In addition, some workflow patterns may also need to use the Oracle Business Rules Designer to define task routing policies or the list of approvers.

For more information, see [Section 27.3, "Creating the Human Task Definition with the Human Task Editor."](#)

27.2.2 Associate the Human Task Definition with a BPEL Process

You can associate the `.task` file that consists of the human task settings with a BPEL process in Oracle BPEL Designer. Association is made with a human task that you drag into your BPEL process flow for configuring, as shown in [Figure 27-1](#).

Figure 27–1 Dragging a Human Task into a BPEL Process

You also specify the task definition, task initiator, task priority, and task parameter mappings that carry the input data to a BPEL variable. You can also define advanced features, such as the scope and global task variables names (instead of accepting the default names), task owner, identification key, BPEL callback customizations, and whether to extend the human task to include other workflow tasks.

When association is complete, a task service partner link is created. The task service exposes the operations required to act on the task.

You can also create the human task as a standalone component only in the SOA Composite Editor and not associate it with a BPEL process. Standalone human task service components are useful for environments in which there is no need for any automated activity in an application. In the standalone case, the client can create the task themselves.

For more information, see [Section 27.4, "Associating the Human Task Service Component with a BPEL Process."](#)

27.2.3 Generate the Task Form

You can generate a task form using the Oracle Application Development Framework (ADF). This form is used for displaying the task details on which you act at runtime in Oracle BPM Worklist.

For information on generating the task form, see [Chapter 28, "Designing Task Forms for Human Tasks."](#)

27.3 Creating the Human Task Definition with the Human Task Editor

The Human Task Editor enables you to define the metadata for the task. The editor enables you to specify human task settings, such as task outcome, payload structure, assignment and routing policy, expiration and escalation policy, notification settings, and so on.

27.3.1 How to Create a Human Task Service Component

You create a human task service component in the SOA Composite Editor or in Oracle BPEL Designer. After creation, you design the component in the Human Task Editor.

The method by which you create the human task service component determines whether the component can be associated later with a BPEL process service component or is a standalone component in the SOA Composite Editor.

To create a human task service component in the SOA Composite Editor:

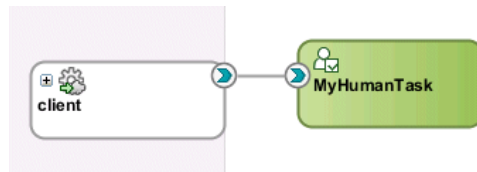
1. Go to the SOA project in which to create a human task service component in the SOA Composite Editor.
2. From the Component Palette list, select **SOA**.
The list refreshes to display service components and service adapters.
3. From the list, drag a **Human Task** into the designer.
The Create Human Task dialog appears.
4. In the **Name** field, enter a name.
The name you enter becomes the `.task` file name.
5. Note the **Create Composite Service with SOAP Bindings** checkbox. The selection of this checkbox determines how the human task service component is created.
 - a. To create a human task service component that you later associate with a BPEL process service component, do *not* select the **Create Composite Service with SOAP Bindings** checkbox. The human task service component is created as a component that you explicitly associate with a BPEL process service component. [Figure 27-2](#) provides details.

Figure 27-2 Human Task Component



- b. To create the human task service component as a standalone component in the SOA Composite Editor, select the **Create Composite Service with SOAP Bindings** checkbox. This creates a human task service component that is automatically wired to a Simple Object Access Protocol (SOAP) web service. [Figure 27-3](#) provides details.

Figure 27-3 Standalone Human Task Component



This web service provides external customers with an entry point into the human task service component of the SOA composite application.

6. Click **OK**.

To create a human task in Oracle BPEL Designer:

1. In the Component Palette, expand **SOA Components**.

2. From the list, drag a **Human Task** into the designer.

The Create Human Task dialog appears.

3. Click the **Add** icon to create a human task.

4. In the **Name** field, enter a name.

The name you enter becomes the `.task` file name.

5. In the **Title** field, enter a task.

6. Click **OK**.

The Human Task Editor appears.

Note: You can also create a human task that you *later* associate with a BPEL process by selecting **New** from the **File** main menu, then selecting **SOA Tier > Service Components > Human Task**.

For more information about creating a human task service component in the SOA Composite Editor, see [Chapter 2, "Developing SOA Composite Applications with Oracle SOA Suite."](#)

27.3.2 What Happens When You Create a Human Task Service Component

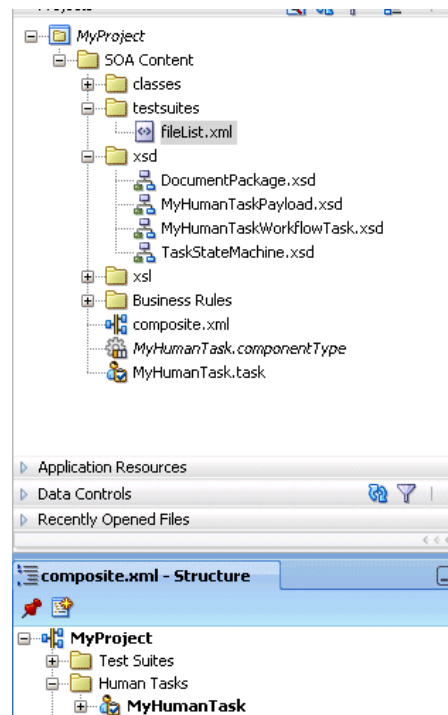
When a human task is created, the following folders and files appear:

- The human task settings specified in the Human Task Editor are saved to a metadata task configuration file in the metadata service (MDS) repository with a `.task` extension. This file appears in the Application Navigator under **SOA_Project_Name > SOA Content**. You can re-edit the settings in this file by double-clicking the following:
 - The `.task` file in the Application Navigator in either the SOA Composite Editor or Oracle BPEL Designer
 - The human task icon in the SOA Composite Editor or in your BPEL process in Oracle BPEL Designer.

This reopens the `.task` file in the Human Task Editor.

- A **Human Tasks** folder containing the human task you created appears in the Structure window of the SOA Composite Editor.

[Figure 27–4](#) shows these folders and files.

Figure 27–4 Human Task Folders and Files

For information about available samples, see [Section 26.3.2, "Designing a Human Task from Start to Finish."](#)

27.3.3 How to Access the Sections of the Human Task Editor

To access the sections of the Human Task Editor:

1. Double-click the **Human Task** icon in the SOA Composite Editor or double-click the Human Task icon in Oracle BPEL Designer and click the **Edit** icon in the upper right corner.

The Human Task Editor consists of the main sections shown on the left side in [Figure 27–5](#). These sections enable you to design the metadata of a human task.

Figure 27–5 Human Task Editor

Instructions for using these main sections of the Human Task Editor to create a workflow task are listed in [Table 27–1](#).

Table 27–1 Human Task Editor

Section	Description	See...
General (title, description, outcomes, category, priority, owner, and application context)	Enables you to define task details such as title, task outcomes, owner, and other attributes.	Section 27.3.4, "How to Specify the Title, Description, Outcome, Priority, Category, Owner, and Application Context"
Data	Enables you to define the structure (message elements) of the task payload (the data in the task).	Section 27.3.5, "How to Specify the Task Payload Data Structure"
Assignment	Enables you to assign participants to the task and create a policy for routing the task through the workflow.	Section 27.3.6, "How to Assign Task Participants" Section 27.3.7, "How to Select a Routing Policy"
Presentation	Enables you to specify the following settings: ■ Multilingual settings ■ WordML and custom style sheets for attachments	Section 27.3.8, "How to Specify Multilingual Settings and Style Sheets"
Deadlines	Enables you to specify the expiration duration of a task, custom escalation Java classes, and due dates.	Section 27.3.9, "How to Escalate, Renew, or End the Task"
Notification	Enables you to create and send notifications when a user is assigned a task or informed that the status of the task has changed.	Section 27.3.10, "How to Specify Participant Notification Preferences"
Access	Enables you to specify access rules for task content and task actions, workflow signature policies, and assignment restrictions.	Section 27.3.11, "How to Specify Access Policies and Task Actions on Task Content" Section 27.3.12, "How to Specify a Workflow Digital Signature Policy" Section 27.3.13, "How to Specify Restrictions on Task Assignments"
Events	Enables you to specify callback classes and task and routing assignments in BPEL callbacks.	Section 27.3.14, "How to Specify Java or Business Event Callbacks"

27.3.4 How to Specify the Title, Description, Outcome, Priority, Category, Owner, and Application Context

To specify the title, description, outcome, priority, category, owner, and application context:

1. Click the **General** tab.

[Figure 27–6](#) shows the **General** section of the Human Task Editor.

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

Figure 27–6 Human Task Editor — General Section

General

Data

Assignment

Presentation

Deadlines

Notification

Access

Events

Task Title: Plain Text

Description:

Outcomes:

Priority: 3 (Normal)

Category: By expression

Owner: User Static ☐

Application Context:

Instructions for configuring the following subsections of the **General** section are listed in [Table 27–2](#):

Table 27–2 Human Task Editor — General Section

For This Subsection...	See...
Title	Section 27.3.4.1, "Specifying a Task Title"
Description	Section 27.3.4.2, "Specifying a Task Description"
Outcomes	Section 27.3.4.3, "Specifying a Task Outcome"
Priority	Section 27.3.4.4, "Specifying a Task Priority"
Category	Section 27.3.4.5, "Specifying a Task Category"
Owner	Section 27.3.4.6, "Specifying a Task Owner"
Application Context	Section 27.3.4.7, "Specifying an Application Context"

27.3.4.1 Specifying a Task Title

To specify a task title:

Enter an optional task title. The title defaults to this value only if the initiated task does not have a title set in it. The title provides a visual identifier for the task. The task title displays in Oracle BPM Worklist. You can also search on titles in Oracle BPM Worklist.

- In the **Task Title** field of the **General** section, select a method for specifying a task title:
 - Plain Text:** Manually enter a name (for example, Vacation Request Approved).
 - Text and XPath:** Enter a combination of manual text and a dynamic expression. After manually entering a portion of the title (for example, Approval Required for Order Id:), place the cursor one blank space to the right of the text and click the icon to the right of this field. This displays the Expression Builder for dynamically creating the remaining portion of the title. After completing the dynamic portion of the name, click **OK** to return to this field. The complete name is displayed. For example:

Approval Required for Order Id: <%/task:task/task:payload/task:orderId%>

The expression is resolved during runtime with the exact order ID value from the task payload.

If you enter a title in the **Task Title** field of the **General** tab of the Create Human Task dialog described in [Section 27.4.3.1, "Specifying the Task Title,"](#) the title you enter here is overridden.

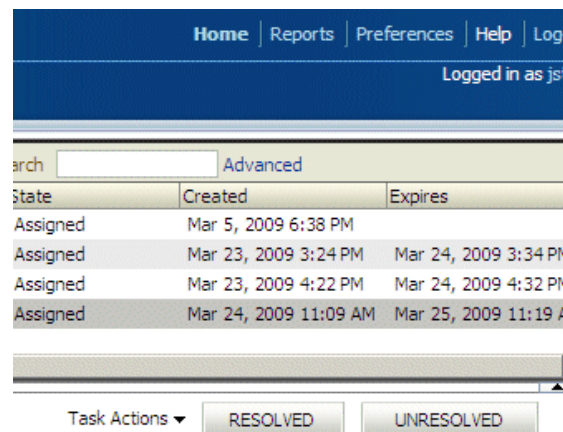
27.3.4.2 Specifying a Task Description

You can optionally specify a description of the task in the **Description** field of the **General** section. The description enables you to provide additional details about a task. For example, if the task title is `Computer Upgrade Request`, you can provide additional details in this field, such as the model of the computer, amount of CPU, amount of RAM, and so on. The description does not display in Oracle BPM Worklist.

27.3.4.3 Specifying a Task Outcome

Task outcomes capture the possible outcomes of a task. Oracle BPM Worklist displays the outcomes you specify here as the possible task actions to perform during runtime. [Figure 27-7](#) provides details.

Figure 27-7 Outcomes in Oracle BPM Worklist



You can specify the following types of task outcomes:

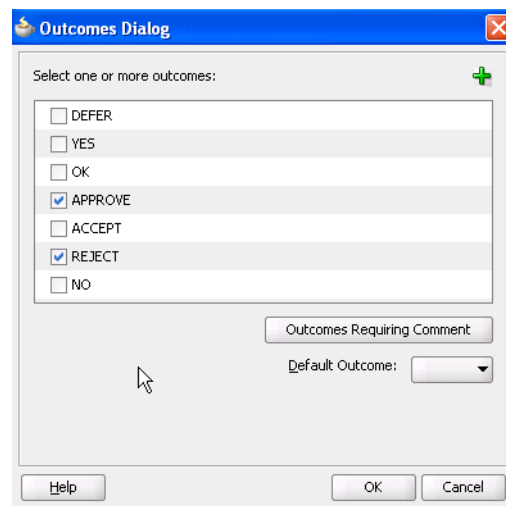
- Select a seeded outcome
- Enter a custom outcome

The task outcomes can also have runtime display values that are different from the actual outcome value specified here. This permits outcomes to be displayed in a different language in Oracle BPM Worklist. For more information about internationalization, see [Section 27.3.8.2, "Specifying Multilingual Settings."](#)

To specify a task outcome:

1. To the right of the **Outcomes** field in the **General** section, click the **Search** icon.

The Outcomes dialog shown in [Figure 27-8](#) displays the possible outcomes for tasks. **APPROVE** and **REJECT** are selected by default.

Figure 27–8 Outcomes Dialog

2. Enter the information shown in [Table 27–3](#).

Table 27–3 Outcomes Dialog

Field	Description
Select one or more outcomes	Select additional task outcomes or deselect the default outcomes.
Add icon	Click to invoke a dialog for adding custom outcomes. In the Name field of the dialog, enter a custom name, and click OK. Your outcome displays in the Outcomes field. Notes: Be aware of the following naming restrictions: ■ Do not specify a custom name that matches a name listed in the Actions tab of the Access section of the Human Task Editor (for example, do not specify <code>Delete</code>). Specifying the same name can cause problems at runtime. ■ Do not specify a custom name with blank spaces (for example, <code>On Hold</code>). This causes an error when the custom outcome is accessed in Oracle BPM Worklist. If you need to specify an outcome with spaces, use a resource bundle. For more information, see Chapter 31, "Introduction to Human Workflow Services." ■ A custom task outcome must begin with a letter of the alphabet, either upper or lower case. It should contain only letters of the alphabet and the numbers zero (0) through nine (9).
Outcomes Requiring Comment	Click to select an outcome to which an assignee adds comments in Oracle BPM Worklist at runtime. The assignee must add the comments and perform the action without saving the task at runtime.
Default Outcome	Select the default outcome for this outcome.

The seeded and custom outcomes selected here display for selection in the **Majority Voted Outcome** section of the parallel participant type.

3. For more information, see [Section 27.3.6.2.1, "Specifying the Voting Outcome."](#)

27.3.4.4 Specifying a Task Priority

Specify the priority of the tasks. Priority can be **1** through **5**, with **1** being the highest. By default, the priority of a task is **3**. This priority value is overridden by any priority value you select in the **General** tab of the Create Human Task dialog. You can filter tasks based on priority and create views on priorities in Oracle BPM Worklist.

To specify a task priority:

1. From the **Priority** list in the **General** section, select a priority for the task.

For more information about specifying a priority value in the Create Human Task dialog, see [Section 27.4.3.2, "Specifying the Task Initiator and Task Priority."](#)

27.3.4.5 Specifying a Task Category

You can optionally specify a task category in the **Category** field of the **General** section. This categorizes tasks created in a system. For example, in a help desk environment, you may categorize customer requests as either software-related or hardware-related. The category displays in Oracle BPM Worklist. You can filter tasks based on category and create views on categories in Oracle BPM Worklist.

To specify a task category:

1. Select a method for specifying a task category in the **Category** field of the **General** section:
 - **By Name:** Manually enter a name.
 - **By Expression:** Click the icon to the right of this field to display the Expression Builder for dynamically creating a category.

27.3.4.6 Specifying a Task Owner

The task owner can view the tasks belonging to business processes they own and perform operations on behalf of any of the assigned task participant types. Additionally, the owner can also reassign, withdraw, or escalate tasks. The task owner can be considered the business administrator for a task. The task owner can also be specified in the **Advanced** tab of the Create Human Task dialog described in [Section 27.4.4.2, "Specifying a Task Owner."](#) The task owner specified in the **Advanced** tab overrides any task owner you enter here.

For more information about the task owner, see [Section 26.2.1.3, "Task Stakeholders."](#)

To specify a task owner:

1. Select a method for specifying the task owner:
 - Statically through the identity service user directory or the list of application roles
 - Dynamically through an XPath expression

For example:

- If the task has a payload message attribute named `po` within which the owner is stored, you can specify an XPath expression such as:
`/task:task/task:payload/po:purchaseOrder/po:owner`
- `ids:getManager('jstein', 'jazn.com')`

The manager of `jstein` is the task owner.

For more information about users, groups, and application roles, see [Section 26.2.1.1.3, "Participant Assignment."](#)

27.3.4.6.1 Specifying a Task Owner Statically Through the User Directory or a List of Application Roles

Task owners can be selected by browsing the user directory (Oracle Internet Directory, Java AuthoriZationN (JAZN)/XML, LDAP, and so on) or a list of application roles configured for use with Oracle SOA Suite.

To specify a task owner statically through the user directory or a list of application roles:

1. In the first list to the right of the **Owner** field in the **General** section, select **User**, **Group**, or **Application Role** as the type of task owner. [Figure 27–9](#) provides details.

Note: By default, group names in human tasks are case sensitive. Therefore, if you select **Group** and enter a name in upper case text (for example, LOANAGENTGROUP), no task is displayed under the **Administrative Tasks** tab in Oracle BPM Worklist. To enable group names to be case agnostic (case insensitive), you must set the **caseSensitiveGroups** property to `false` in the System MBeans Browser. For information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

Figure 27–9 Specify a Task Owner By Browsing the User Directory or Application Roles

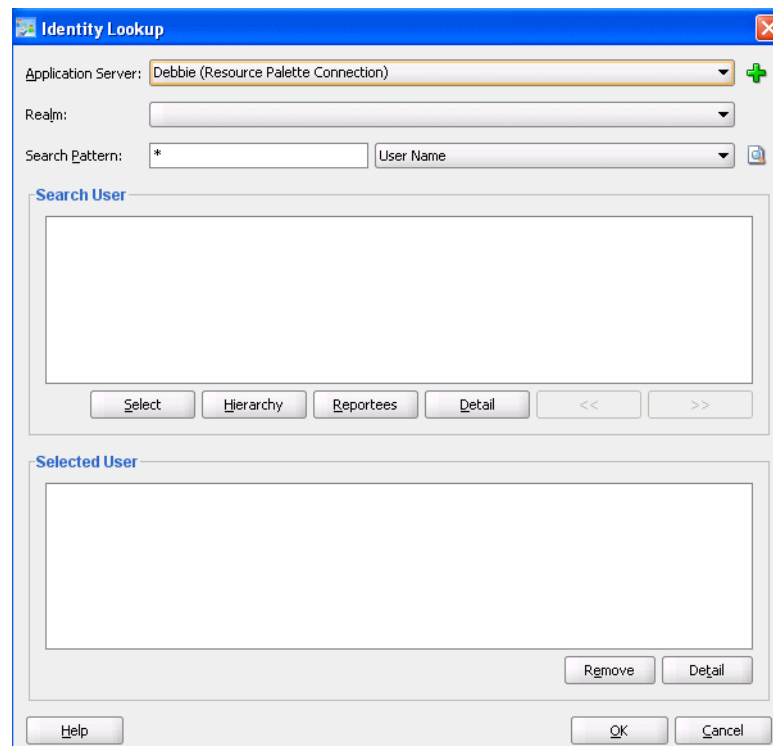


2. In the second list to the right of the **Owner** field in the **General** section, select **Static**.
3. See the step in [Table 27–4](#) based on the type of owner you selected.

Table 27–4 Type of Owner

If You Selected...	See Step...
User or Group	4
Application Role	5

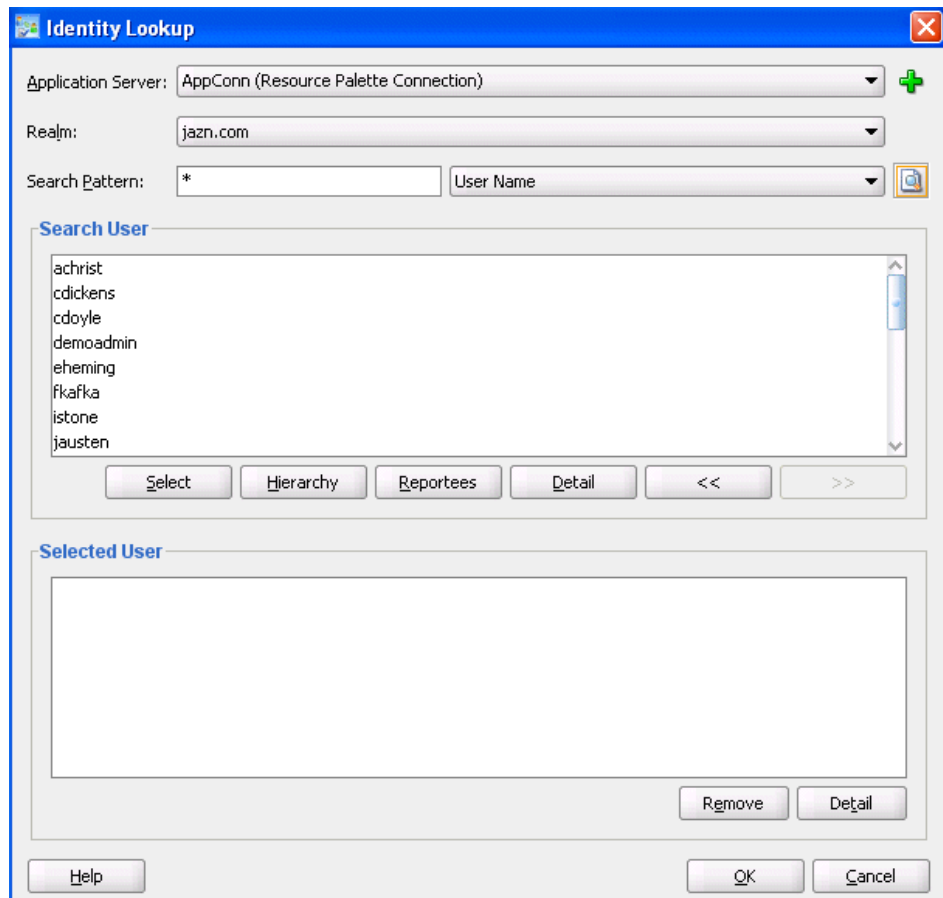
4. If you selected **User** or **Group**, the Identity Lookup dialog shown in [Figure 27–10](#) appears.

Figure 27–10 Identity Lookup Dialog

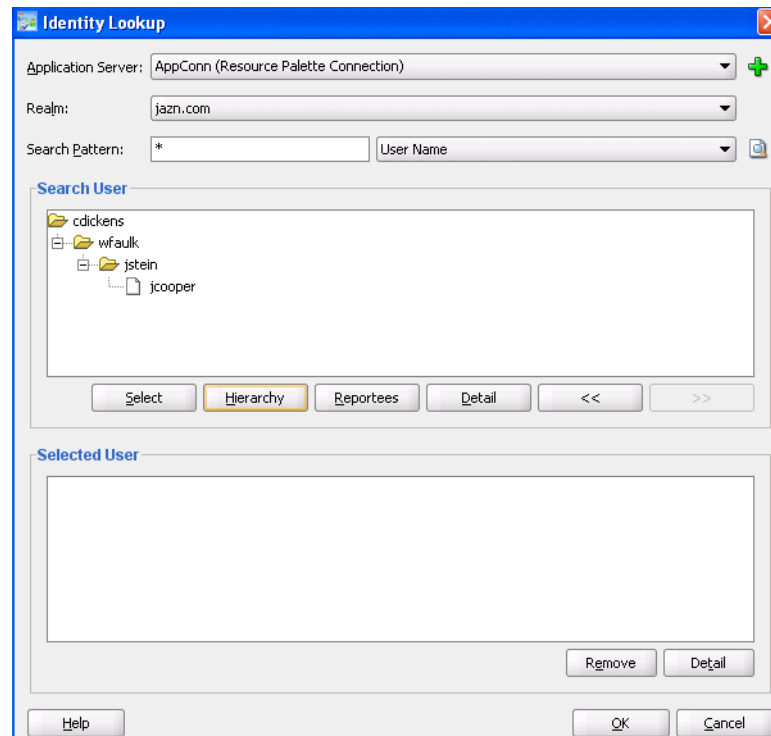
To select a user or group, you must first create an application server connection by clicking the **Add** icon. Note the following restrictions:

- Do not create an application server connection to an Oracle WebLogic Administration Server from which to retrieve the list of identity service realms. This is because there is no identity service running on the Administration Server. Therefore, no realm information displays and no users display when performing a search with a search pattern in the Identity Lookup dialog. Instead, create an application server connection to a managed Oracle WebLogic Server.
- You must select an application server connection configured with the complete domain name (for example, `myhost.us.oracle.com`). If you select a connection configured only with the hostname (for example, `myhost`), the **Realm** list may not display the available realms. If the existing connection does not include the domain name, perform the following steps:
 - In the **Resource Palette**, right-click the application server connection.
 - Select **Properties**.
 - In the **Configuration** tab, add the appropriate domain to the hostname.
 - Return to the Identity Lookup dialog and reselect the connection.
- a. Select or create an application server connection to display the realms for selection. A realm provides access to a policy store of users and roles (groups).
- b. Search for the owner by entering a search string such as `jcooper`, `j*`, `*`, and so on. Clicking the **Lookup** icon to the right of the **User Name** field fetches all the users that match the search criteria. [Figure 27–11](#) provides details. One or more users or groups can be highlighted and selected by clicking **Select**.

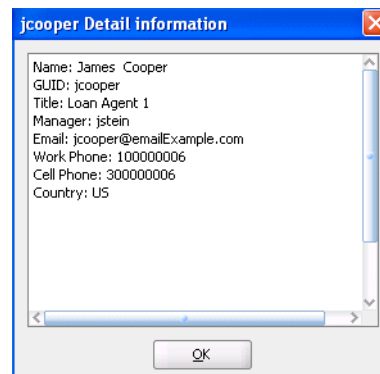
Figure 27–11 Identity Lookup with Realm Selected



- c. View the hierarchy of a user by highlighting the user and clicking **Hierarchy**. Similarly, clicking **Reportees** displays the reportees of a selected user or group. [Figure 27–12](#) provides details.

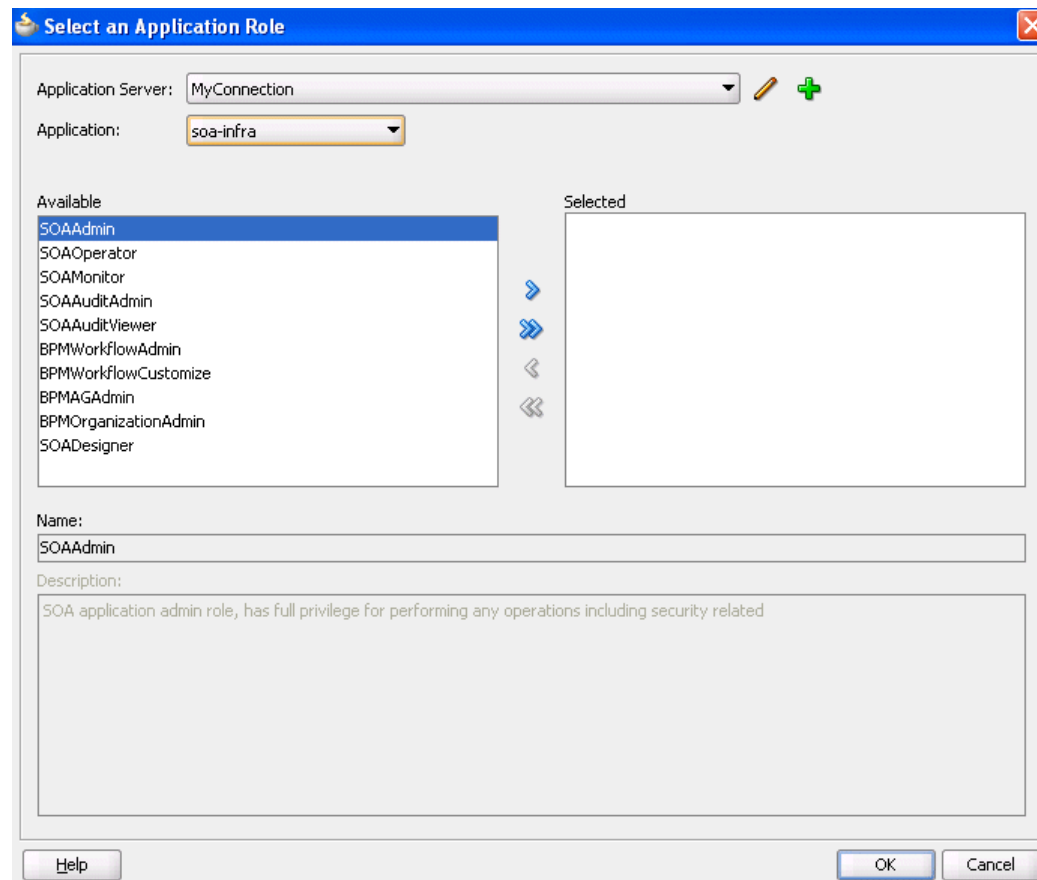
Figure 27–12 User Hierarchy in Identity Lookup Dialog

- d. View the details of a user or group by highlighting the user or group and clicking **Detail**. Figure 27–13 provides details.

Figure 27–13 User or Group Details

- e. Click **OK** to return to the Identity Lookup dialog.
 - f. Click **Select** to add the user to the **Selected User** section.
 - g. Click **OK** to return to the Human Task Editor.
Your selection displays in the **Owner** field.
5. If you selected **Application Role**, the Select an Application Role dialog appears.
 - a. In the **Application Server** list, select the type of application server that contains the application role or click the **Add** icon to launch the Create Application Server Connection wizard to create a connection.

- b. In the **Application** list, select the application that contains the application roles (for example, a custom application or **soa-infra** for the SOA Infrastructure application).
- c. In the **Available** section, select appropriate application roles and click the > button. To select all, click the >> button. [Figure 27–14](#) provides details.

Figure 27–14 Application Role

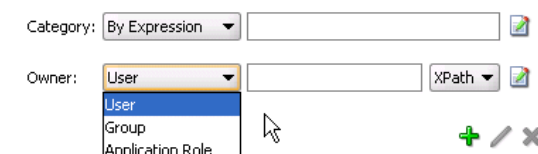
- d. Click OK.

27.3.4.6.2 Specifying a Task Owner Dynamically Through an XPath Expression

Task owners can be selected dynamically in the Expression Builder dialog.

To specify a task owner dynamically:

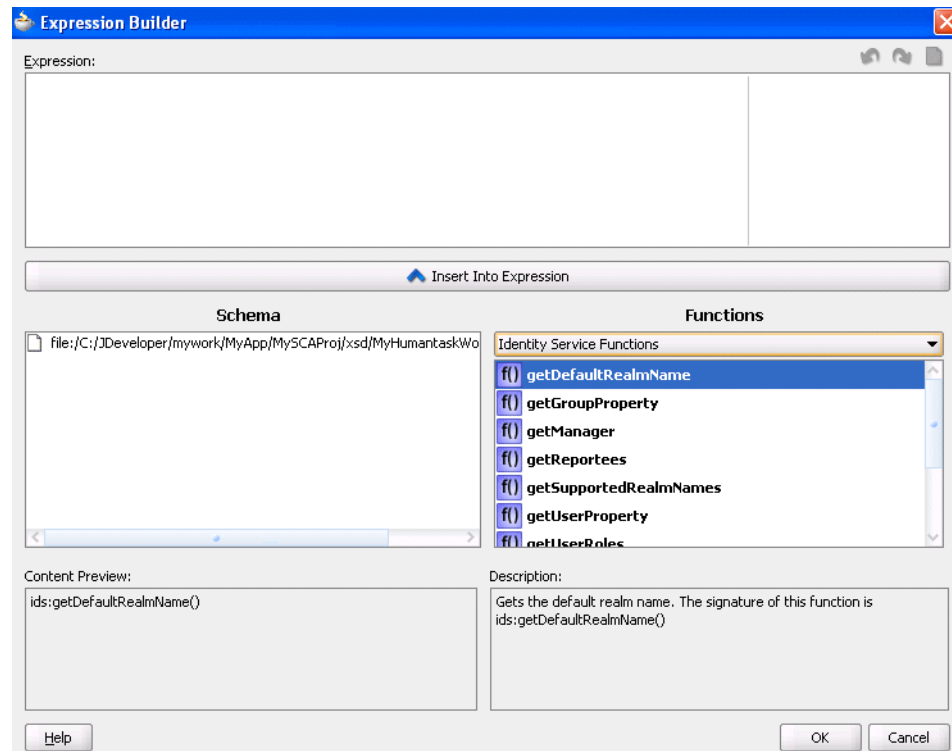
1. In the first list to the right of the **Owner** field in the **General** section, select **User**, **Group**, or **Application Role** as the type of task owner. [Figure 27–15](#) provides details.

Figure 27–15 Specify a Task Owner Dynamically

2. In the second list to the right of the **Owner** field in the **General** section, select **XPath**.
3. Click the icon to launch the Expression Builder.

This displays the Expression Builder dialog shown in [Figure 27–16](#):

Figure 27–16 Expression Builder



4. Browse the available variable schemas and functions to create a task owner.
5. Click **OK** to return to the Human Task Editor.

Your selection displays in the **Owner** field.

For more information, see the following:

- Click **Help** for instructions on using the Expression Builder dialog and XPath Building Assistant
- [Appendix B, "XPath Extension Functions"](#) for information about workflow service dynamic assignment functions, identity service functions, and instructions on using the XPath Building Assistant

27.3.4.7 Specifying an Application Context

You can specify the name of the application that contains the application roles used in the task. This indicates the context in which the application role operates. If you do not explicitly create a task, but end up having one, you can set up the context.

1. In the **Application Context** field of the **General** section, enter the name.

27.3.5 How to Specify the Task Payload Data Structure

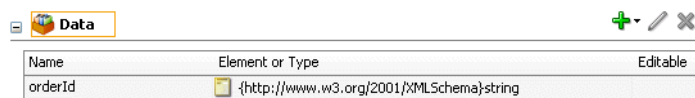
[Figure 27–17](#) shows the **Data** section of the Human Task Editor.

This section enables you to specify the structure (message elements) of the task payload (the data in the task) defined in the XSD file. You create parameters to represent the elements in the XSD file. This makes the payload data available to the workflow task. For example:

- You create a parameter for an order ID element for placing an order from a store front application.
- You create parameters for the location, type, problem description, severity, status, and resolution elements for creating a help desk request.

Task payload data consists of one or more elements or types. Based on your selections, an XML schema definition is created for the task payload.

Figure 27–17 Human Task Editor — Parameters Section

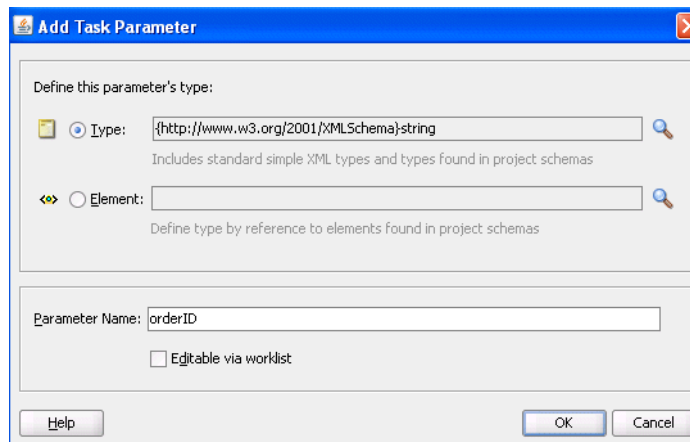


To specify the task payload data structure:

1. Click the **Data** tab.
2. Click the **Add** icon and select a payload type:
 - String
 - Integer
 - Boolean
 - Other

The Add Task Parameter dialog is displayed, as shown in [Figure 27–18](#).

Figure 27–18 Add Task Parameter Dialog



3. Enter the details described in [Table 27–5](#):

Table 27–5 Add Task Parameter Dialog Fields and Values

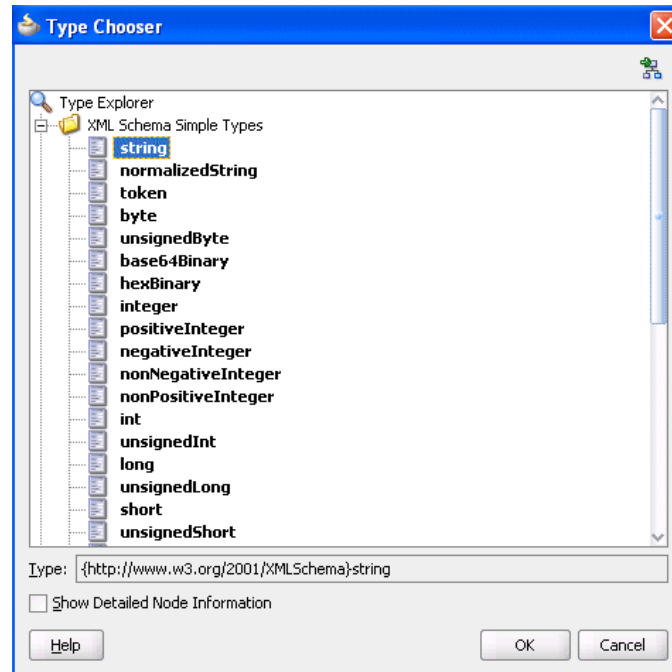
Field	Description
Parameter Type	Select Type or Element and click the Search icon to display the Type Chooser dialog for selecting the task parameter.

Table 27–5 (Cont.) Add Task Parameter Dialog Fields and Values

Field	Description
Parameter Name	Accept the default name or enter a custom name. This field only displays if Type is the selected parameter type.
Editable via worklist	Select this checkbox to enable users to edit this part of the task payload in Oracle BPM Worklist. For example, for a loan approval task, the APR attribute may need to be updated by the user reviewing the task, but the SSN field may not be editable. You can also specify access rules that determine the parts of a task that participants can view and update. For more information, see Section 27.3.11.1, "Specifying Access Policies on Task Content."

Note: You can only define payload mapped attributes (previously known as flex field mappings) in Oracle BPM Worklist for payload parameters that are simple XML types (string, integer, and so on) or complex types (for example, a purchase order, and so on). If you must search tasks using keywords or define views or delegation rules based on task content, then you must use payload parameters based on simple XML types. These simple types can be mapped to flex columns in Oracle BPM Worklist.

4. Select the type, as shown in [Figure 27–19](#).

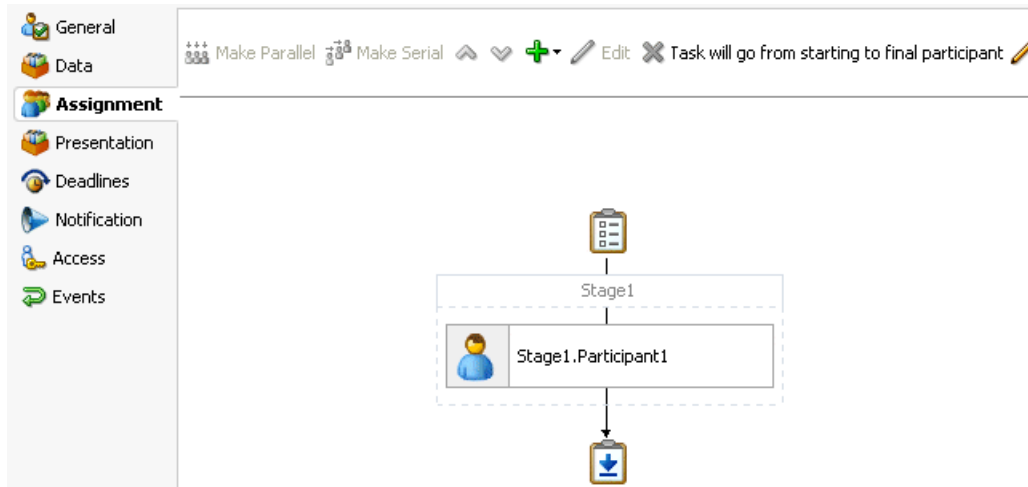
Figure 27–19 Parameter Type

5. Click **OK** to return to the Human Task Editor.
Your selection displays in the **Data** section.
6. To edit your selection, select it and click the **Edit** icon in the upper right part of the **Data** section.

27.3.6 How to Assign Task Participants

Figure 27–20 shows the **Assignment** section of the Human Task Editor. This section enables you to select a participant type that meets your business requirements. While configuring the participant type, you build lists of users, groups, and application roles to act upon tasks.

Figure 27–20 Human Task Editor — Assignment Section



You can easily mix and match participant types to create simple or complex workflow routing policies. You can also extend the functionality of a previously configured human task to model more complex workflows.

A participant type is grouped in a block under a stage (for example, named **Stage1** in Figure 27–20). A stage is a way of organizing the approval process for blocks of participant types. You can have one or more stages in sequence or in parallel. Within each stage, you can have one or more participant type blocks in sequence or in parallel. The up and down keys enable you to rearrange the order of your participant type blocks.

For example:

- You can create all participant type blocks in a single stage (for example, a purchase order request in which the entire contents of the order are approved or rejected as a whole).
- You can create more complex approval tasks that may include one or more stages. For example, you can place one group of participant type blocks in one stage and another block in a second stage. The list of approvers in the first stage handles line entry approvals and the list of approvers in the second stage handles header entry approvals.

Each of the participant types has an associated editor that you use for configuration tasks. The sequence in which the assignees are added indicates the execution sequence.

To specify a different stage name or have a business requirement that requires you to create additional stages, perform the following steps. Note that creating additional stages is an advanced requirement that may not be necessary for your environment.

For more information about participant types, see [Section 26.2.1.1, "Task Assignment and Routing."](#)

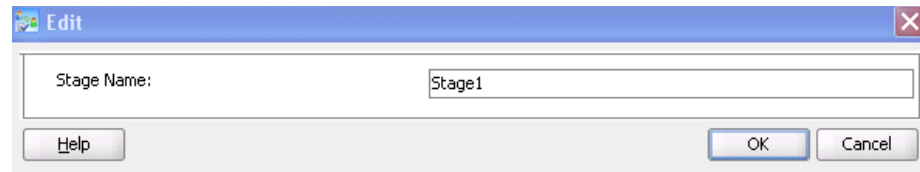
To specify a stage name and add parallel and sequential blocks:

The stage is named **Stage1** by default. If you want, you can change the name.

1. Double-click the name.

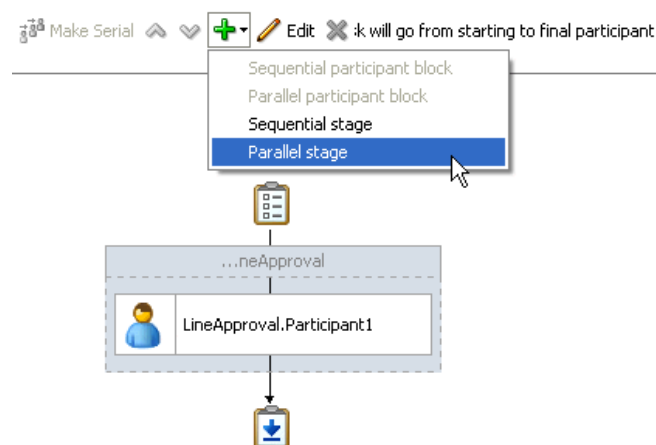
The Edit dialog shown in [Figure 27-21](#) appears.

Figure 27-21 Edit Dialog



2. Enter a name, and click **OK**.
3. Select the stage and its participant type block, as shown in [Figure 27-22](#).
4. Click the **Add** icon.

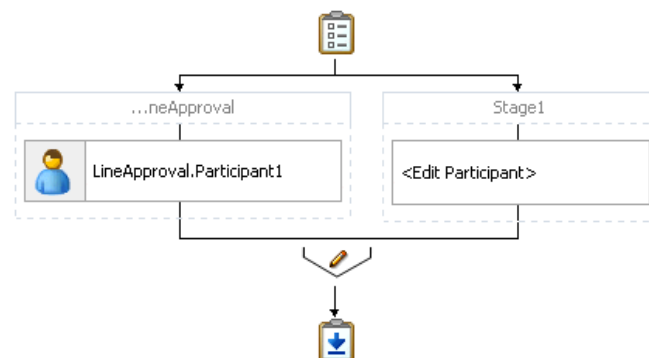
Figure 27-22 Add a Second Stage



5. Select an option from the list (for example, **Parallel stage**).

A second stage is added in parallel to the first stage, as shown in [Figure 27-23](#).

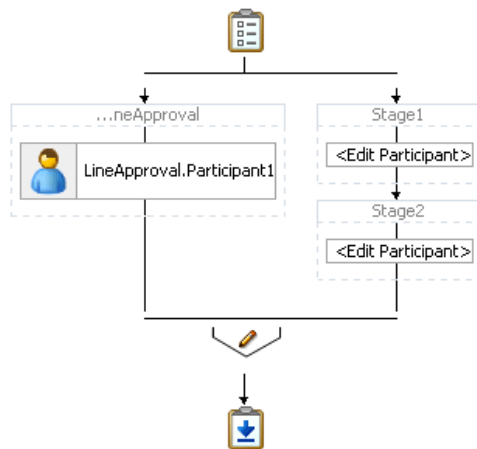
Figure 27-23 Parallel Stage



6. Select the second stage on the right, and click the **Add** icon. Note that if you do not select the second stage (for this example, named **Stage1** in Figure 27–24) and instead select only the participant type block (for example, named **Edit Participant** in Figure 27–24), all options under the **Add** icon are disabled.
7. Select **Sequential stage**.

A sequential stage is added below the selected block.

Figure 27–24 Sequential Stage



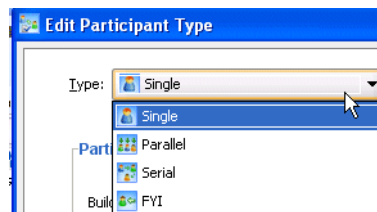
You create participant types within these blocks.

To assign task participants:

1. In the **Assignment** section, perform one of the following tasks:
 - a. Highlight the block below the stage box and click the **Edit** icon. The first time you create a task participant, the box is labeled **<Edit Participant>**.
 - or
 - b. Double-click the participant box below the stage box.

The Edit Participant Type dialog appears. This dialog enables you to select a specific participant type.
2. From the **Type** list, select a participant type shown in Figure 27–25.

Figure 27–25 Type List



3. See the section shown in Table 27–6 based on your selection.

Table 27–6 Participant Types

Participant Type	For a Description of this Participant Type, See...	For Instructions on Configuring this Participant Type, See...
■ Single	Section 26.2.1.1.2, "Participant Type"	Section 27.3.6.1, "Configuring the Single Participant Type"
■ Parallel		Section 27.3.6.2, "Configuring the Parallel Participant Type"
■ Serial		Section 27.3.6.3, "Configuring the Serial Participant Type"
■ FYI		Section 27.3.6.4, "Configuring the FYI Participant Type"

27.3.6.1 Configuring the Single Participant Type

[Figure 27–26](#) displays the Edit Participant Type dialog for the single participant type.

Figure 27–26 Edit Participant Type — Single Type

Type: Single Label: Stage1.Participant1
e.g., Approval Manager

Participant List

Build a list of participants using: Names and expressions

Specify attributes using: ☒ Value-based ☐ Rule-based

Participant Names		
Identification Type	Data Type	Value

Requires action from one of the specified participants

Advanced

☐ Limit allocated duration to:

Day Hour Minutes

Over all task duration is currently set to {0} days, {1} hours, and {2} minutes.

☐ Allow this participant to invite other participants

☐ Specify skip rule

To configure the single participant type:

1. In the **Label** field, enter a recognizable label for this participant. This label must be unique among all the participants in the task definition (for example, Approval Manager, Primary Reviewers, and so on).

Instructions for configuring the following subsections of the Edit Participant Type dialog for the single participant type are listed in [Table 27–7](#):

Table 27–7 Edit Participant Type — Single Type

For This Subsection...	See...
Participant List	Section 27.3.6.1.1, "Creating a Single Task Participant List"
Limit allocated duration to (under the Advanced section)	Section 27.3.6.1.2, "Specifying a Time Limit for Acting on a Task"
Allow this participant to invite other participants (under the Advanced section)	Section 27.3.6.1.3, "Inviting Additional Participants to a Task"
Specify skip rule (under the Advanced section)	Section 27.3.6.1.4, "Bypassing a Task Participant"

27.3.6.1.1 Creating a Single Task Participant List

Users assigned to the list of participants can act upon tasks. In this type of assignment list, only one user is required to act on the task. You can provide either a single user or a list of users, groups, or application roles for this pattern. If a list is specified, then all users are assigned the task; one of them must acquire and act upon the task. When one user acts on it, the task is withdrawn from the task list of other assignees.

You can create several types of lists for the single user participant (and also for the parallel, serial, and FYI user participants):

- Value-based name and expression lists

These lists enable you to statically or dynamically select users, groups, or application roles as task assignees.

- Value-based management chain lists

Management chains are typically used for serial approvals through multiple users in a management chain hierarchy. Therefore, this list is most likely useful with the serial participant type. This is typically the case if you want all users in the hierarchy to act upon the task. Management chains can also be used with the single participant type. In this case, however, all users in the hierarchy get the task assigned at the same time. As soon as one user acts on the task, it is withdrawn from the other users.

For example, a purchase order is assigned to a manager. If the manager approves the order, it is assigned to their manager. If that manager approves it, it is assigned to their manager, and so on until three managers approve the order. If any managers reject the request or the request expires, the order is rejected if you specify an abrupt termination condition. Otherwise, the task flow continues to be routed.

- Rule-based names and expression lists and management chain lists

Business rules enable you to create the list of task participants with complex expressions. For example, you create a business rule in which a purchase order request below \$5000 is sent to a manager for approval. However, if the purchase order request exceeds \$5000, the request is sent to the manager of the manager for approval. Two key features of business rules are facts and action types, which are described in [Section 27.3.7.2, "Specifying Advanced Task Routing Using Business Rules."](#)

When you select a participant type, the dialog that displays enables you to choose an option for building your list of task participant assignees (users, groups, or application

roles), as shown in [Figure 27-27](#). The three selections described above are available: **Names and expressions**, **Management Chain**, and **Rule-based**.

Figure 27-27 Build a List of Participants

Edit Participant Type

Type: Single Label: Stage1.Participant2
e.g., Approval Manager

Participant List

Build a list of participants using: Names and expressions

Specify attributes using: ☒ Value-based ☐ Rule-based

Participant Names		
Identification Type	Data Type	Value

After selecting an option, you dynamically assign task participant assignees (users, groups, or application roles) and a data type, as shown in [Figure 27-28](#).

Figure 27-28 Assignment of Task Assignees

Edit Participant Type

Type: Single Label: Stage1.Participant2
e.g., Approval Manager

Participant List

Build a list of participants using: Names and expressions

Specify attributes using: ☒ Value-based ☐ Rule-based

Participant Names		
Identification Type	Data Type	Value
Group	By Name	

This section describes how to create these lists of participants.

Creating Participant Lists Consisting of Value-Based Names and Expressions

Select a method for statically or dynamically assigning a user, group, or application role as a task participant.

For conceptual information about the following:

- Users, groups, or application roles, see [Section 26.2.1.1.3, "Participant Assignment."](#)
- Statically and dynamically assigning task participants, see [Section 26.2.1.2, "Static, Dynamic, and Rule-Based Task Assignment."](#)

To create participant lists consisting of value-based names and expressions:

1. From the **Build a list of participants using** list, select **Names and expressions**.
2. From the **Specify attributes using** list, select **Value-based**.

The dialog refreshes to display the fields shown in [Figure 27–29](#).

Figure 27–29 Value-Based Names and Expressions

Participant List

Build a list of participants using: Names and expressions ▼

Specify attributes using: ☒ Value-based ☐ Rule-based

Identification Type	Data Type	Value
User	By Name	jstein

3. Click the **Add** icon and select a user, group, or application role as a task participant.
4. To change your selection in the **Identification Type** column, click it to invoke a dropdown list.
5. In the **Data Type** column, click your selection to invoke a dropdown list to assign a value:

- **By Name:** If your identification type is a user or group, click the **Browse** icon (the dots) on the right to display a dialog for selecting a user or group configured through the identity service. The identity service enables the lookup of user properties, roles, and group memberships. User information is obtained from an LDAP server such as Oracle Internet Directory. You can use wild cards (*) to search for IDs.

If your selection is an application role, click the **Browse** icon to display the Select an Application Role dialog for selecting an application role. To search for application roles, you must first create a connection to the application server. When searching, you must specify the application name to find the name of the role. Note that the task definition can refer to only one application name. You cannot use application roles from different applications as assignees or task owners.

- **By Expression:** For a user, group, or application role, click the **Browse** icon to dynamically select a task assignee in the Expression Builder dialog. Use the `bpws:getVariableData(...)` expression or the `ids:getManager()` XPath function.

The **Value** column displays the value you specified.

6. To manually enter a value, click the field in the **Value** column and specify a value.

Creating Participant Lists Consisting of Value-Based Management Chains

Select a method for statically or dynamically assigning management chain parameters as task participants.

For conceptual information about the following:

- Users, groups, or application roles, see [Section 26.2.1.1.3, "Participant Assignment."](#)
- Statically and dynamically assigning task participants, see [Section 26.2.1.2, "Static, Dynamic, and Rule-Based Task Assignment."](#)
- Management chains, see [Section 27.3.6.1.1, "Creating a Single Task Participant List."](#)

To specify participant lists based on value-based management chains:

1. From the **Build a list of participants using** list, select **Management Chain**.
2. From the **Specify attributes using** list, select **Value-based**.

The dialog refreshes to display the fields shown in [Figure 27–30](#).

Figure 27–30 Value-Based Management Chains

Participant List

Build a list of participants using: Management Chain ▾

Specify attributes using: ☒ Value-based ☐ Rule-based

Starting Participant:		
Identification Type	Data Type	Value

Top Participant: By Title ▾

Number of Levels: By Number ▾

3. See Step 3 through Step 6 of [Section 27.3.6.1.1, "Creating a Single Task Participant List"](#) for instructions on assigning a user, group, or application role to a list in the **Starting Participant** table.
4. In the **Top Participant** list, select a method for assigning the number of task participant levels:
 - **By Title:** Select the title of the last (highest) approver in the management chain.
 - **XPath:** Select to dynamically enter a top participant through the Expression Builder dialog.
5. In the **Number of Levels** list, select a method for assigning a top participant:
 - **By Number:** Enter a value for the number of levels in the management chain to include in this task. For example, if you enter 2 and the task is initially assigned to user `jcooper`, both the user `jstein` (manager of `jcooper`) and the user `wfaulk` (manager of `jstein`) are included in the list (apart from `jcooper`, the initial assignee).
 - **XPath:** Select to dynamically enter a value through the Expression Builder dialog.

Creating Participant Lists Consisting of Rulesets

A ruleset provides a unit of execution for rules and for decision tables. In addition, rulesets provide a unit of sharing for rules; rules belong to a ruleset. Multiple rulesets can be executed in order. This is called rule flow. The ruleset stack determines the order. The order can be manipulated by rule actions that push and pop rulesets on the stack. In rulesets, the priority of rules applies to specify the order of firing of rules in the ruleset. Rulesets also provide an effective date specification that identifies that the ruleset is always active, or that the ruleset is restricted based on a time and date range, or a starting or ending time and date.

The method by which you create a ruleset is based on how you access it. This is described in the following section.

To specify participant lists based on rulesets:

Business rules can define the participant list. There are two options for using business rules:

- Rules define parameters of a specific list builder (such as **Names and Expressions** or **Management Chain**). In this case, the task routing pattern is modeled to use a specific list builder. In the list builder, the parameters are listed as coming from rules. Rules return the list builder of the same type as the one modeled in Oracle JDeveloper.
 1. From the **Build a list of participants using** list, select **Names and expressions** or **Management Chain**.
 2. From the **Specify attributes using** list, select **Rule-based**.
 3. In the **List Ruleset** field, enter a ruleset name.

[Figure 27–31](#) provides details.

Figure 27–31 Rulesets

Type: Single Label: Stage1.Participant1
 e.g., Approval Manager

Participant List

Build a list of participants using: Names and expressions

Specify attributes using: ☐ Value-based ☒ Rule-based

List Ruleset: ApprovalGroupRule

4. Click **OK**.
- Rules define the list builder and the list builder parameters. In this case, the list itself is built using rules. The rules define the list builder and the parameters.
 1. From the **Build a list of participants using** list, select **Rule-based**.
 2. In the **List Ruleset** field, enter a ruleset name.

[Figure 27–32](#) provides details.

Figure 27–32 Rulesets

Type: Single Label: Stage1.Participant1
 e.g., Approval Manager

Participant List

Build a list of participants using: Rule-based

List Ruleset: GenericRule

3. Click **OK**.

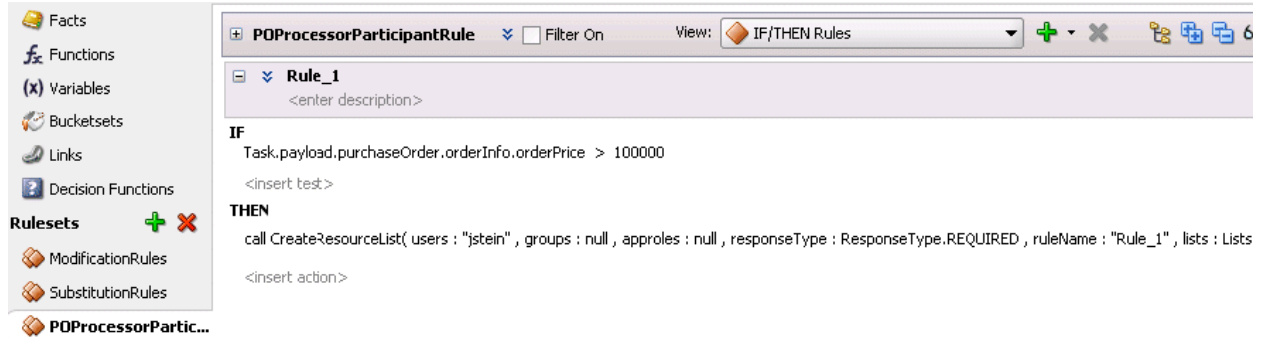
Both options create a rule dictionary, if one is not already created, and preseed several rule functions and facts for easy specifications of the participant list. In the rule dictionary, the following rule functions are seeded to create participant lists:

- `CreateResourceList`
- `CreateManagementChainList`

The Task fact is asserted by the task service for basing rule conditions.

After the rule dictionary is created, the Oracle Business Rules Designer is displayed.

1. Model your rule conditions. In the action part, call one of the above functions to complete building your lists. [Figure 27–33](#) provides details.

Figure 27–33 Business Rules

The parameters for the rule functions are similar to the ones in Oracle JDeveloper modeling. In addition to the configurations in Oracle JDeveloper, some additional options are available in the Oracle Business Rules Designer for the following attributes:

- **responseType**: If the response type is **REQUIRED**, the assignee must act on the task. Otherwise, the assignment is converted to an FYI assignment.
- **ruleName**: The rule name can create reasons for assignments.
- **lists**: This object is a holder for the lists that are built. Clicking this option shows a pre-asserted fact **Lists** object to use as the parameter.

An example of rules specifying management chain-based participants is shown in [Figure 27–34](#).

Figure 27–34 Business Rules

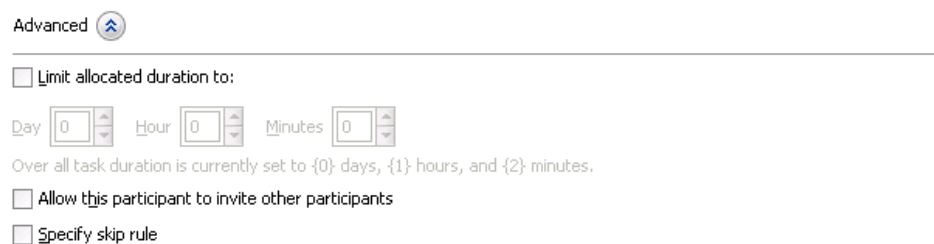
If multiple rules are fired, the list builder created by the rule with the highest priority is selected.

27.3.6.1.2 Specifying a Time Limit for Acting on a Task

You can specify the amount of time a user, group, or application role receives to act on a task. If the user, group, or role does not act in the time specified, the global escalation and renewal policies that you set in the **Deadlines** section (known as the routing slip level) of the Human Task Editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.

To specify a time limit for acting on a task:

1. Expand the **Advanced** section of the Edit Participant Type dialog for the single type, as shown in [Figure 27–35](#).

Figure 27–35 Advanced Section of Edit Participant Type — Single Type

2. Select **Limit allocated duration to**.
3. Specify the amount of time.

For more information about setting the global escalation and renewal policies in the **Deadlines** section of the Human Task Editor, see [Section 27.3.9, "How to Escalate, Renew, or End the Task."](#)

27.3.6.1.3 Inviting Additional Participants to a Task

You can allow a task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow. For example, assume the approval workflow goes from James Cooper to John Steinbeck. If this option is checked, James Cooper can decide to first route it to Irving Stone before it goes to John Steinbeck.

This is also known as ad hoc routing. If this option is selected, **Adhoc Route** is added to the **Actions** list in Oracle BPM Worklist at runtime.

To invite additional participants to a task:

1. Expand the **Advanced** section of the Edit Participant Type dialog for the single type, as shown in [Figure 27–35](#).
2. Select **Allow this participant to invite other participants**.

27.3.6.1.4 Bypassing a Task Participant

You can bypass a task participant (user, group, or application role) if a specific condition is satisfied. For example, if a user submits a business trip expense report that is under a specific amount, no approval is required by their manager.

To bypass a task:

1. Expand the **Advanced** section of the Edit Participant Type dialog for the single type, as shown in [Figure 27–35](#).
2. Select **Specify skip rule**.

This action displays an icon for accessing the Expression Builder dialog for building a condition.

The expression to bypass a task participant must evaluate to a boolean value. For example, `/task:task/task:payload/order:orderAmount < 1000` is a valid XPath expression for skipping a participant.

For more information about creating dynamic rule conditions, see [Section 27.3.7.2, "Specifying Advanced Task Routing Using Business Rules."](#)

27.3.6.2 Configuring the Parallel Participant Type

[Figure 27–36](#) and [Figure 27–37](#) display the upper and lower sections of the Parallel dialog.

This participant type is used when multiple users, working in parallel, must act simultaneously, such as in a hiring situation when multiple users vote to hire or reject an applicant. You specify the voting percentage that is needed for the outcome to take effect, such as a majority vote or a unanimous vote.

For example, a business process collects the feedback from all interviewers in the hiring process, consolidates it, and assigns a hire or reject request to each of the interviewers. At the end, the candidate is hired if the majority of interviewers vote for hiring instead of rejecting.

Figure 27–36 Edit Participant Type — Parallel Type (Upper Section of Dialog)

Type:

Parallel

 Label:

Stage1.Participant1

e.g., Approval Manager

Vote Outcome

A Voted outcome will override the default outcome if the required percentage is reached.
Outcomes will be evaluated in the order listed in the table.

Voted Outcomes	Outcome Type	Value
Any	By Percentage	50

Default Outcome:

☒ Immediately trigger voted outcome when minimum percentage is met

☐ Wait until all votes are in before triggering outcome

☐ Share attachments and comments

Participant List

Build a list of participants using:

Names and expressions

Specify attributes using:

☒ Value-based

☐ Rule-based

Participant Names		
Identification Type	Data Type	Value

Figure 27–37 Edit Participant Type — Parallel Type (Lower Section of Dialog)

☒ Share attachments and comments

Advanced

☒ Limit allocated duration to:

Day

1

Hour

0

Minutes

0

Overall task will never expire.

☐ Allow this participant to invite other participants

☒ Specify skip rule

Skip group vote if condition is satisfied.

To assign participants to the parallel participant type:

1. In the **Label** field, enter a recognizable label for this participant. This label must be unique among all the participants in the task definition (for example, Approval Manager, Primary Reviewers, and so on).

Instructions for configuring the following subsections of the Edit Participant Type dialog for the parallel participant type are listed in [Table 27–8](#):

Table 27–8 Edit Participant Type — Parallel Type

For This Subsection...	See...
Vote Outcome	Section 27.3.6.2.1, "Specifying the Voting Outcome"

Table 27–8 (Cont.) Edit Participant Type — Parallel Type

For This Subsection...	See...
Participant List	Section 27.3.6.2.2, "Creating a Parallel Task Participant List"
Limit allocated duration to (under the Advanced section)	Section 27.3.6.2.3, "Specifying a Time Limit for Acting on a Task"
Allow this participant to invite other participants (under the Advanced section)	Section 27.3.6.2.4, "Inviting Additional Participants to a Task"
Specify skip rule (under the Advanced section)	Section 27.3.6.2.5, "Bypassing a Task Participant"

27.3.6.2.1 Specifying the Voting Outcome You can specify a voted-upon outcome that overrides the default outcome selected in the **Default Outcome** list. This outcome takes effect if the required percentage is reached. Outcomes are evaluated in the order listed in the table.

To specify group voting details:

1. Go to the **Vote Outcome** section of the Edit Participant Type dialog for the parallel type.
2. From the list in the **Voted Outcomes** column, select an outcome for the task (for example, **Any**, **ACCEPT**, **REJECT**, or any other outcome specified in [Section 27.3.4.3, "Specifying a Task Outcome"](#)).

The **Any** outcome enables you to determine the outcome dynamically at runtime. For example, if you select **Any** and set the outcome percentage to 60, then at runtime, whichever outcome reaches 60% becomes the final voted outcome. If 60% of assignees vote to reject the outcome, then it is rejected.

3. From the list in the **Outcome Type** column, select a method for determining the outcome of the final task.
 - **By Expression:** Dynamically specify the details with an XPath expression.
 - **By Percentage:** Specify a percentage value that determines when the outcome of this task takes effect.
4. From the list in the **Value** column, specify a value based on your selection in Step 3.
 - If you selected **By Expression**, click the **Browse** icon to the right of the field to display the Expression Builder dialog for creating an expression.
 - If you selected **By Percentage**, enter a percentage value 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. [Figure 27–38](#) provides details.

Note that this functionality is nondeterministic. For example, selecting a percentage of 30% when there are two subtasks does not make sense.

Figure 27–38 Vote Outcomes Section

Vote Outcome

A Voted outcome will override the default outcome if the required percentage is reached.
Outcomes will be evaluated in the order listed in the table.

Voted Outcomes	Outcome Type	Value
APPROVE	By Percentage	75

5. Click the **Add** icon to specify additional outcomes.
6. In the **Default Outcome** list, select the default outcome or enter an XPath expression for this task to take effect if the consensus percentage value is not satisfied. This happens if there is a tie or if all participants do not respond before the task expires. Seeded and custom outcomes that you entered in the Outcomes dialog in [Section 27.3.4.3, "Specifying a Task Outcome"](#) display in this list.
7. Specify additional group voting details:
 - **Immediately trigger voted outcome when minimum percentage is met**
If selected, the outcome of the task can be computed early with the outcomes of the completed subtasks, enabling the pending subtasks to be withdrawn. For example, assume four users are assigned to act on a task, the default outcome is **APPROVE**, and the consensus percentage is set at **50**. If the first two users approve the task, the third and fourth users do not need to act on the task, since the consensus percentage value has been satisfied.
 - **Wait until all votes are in before triggering outcome**
If selected, the workflow waits for all responses before an outcome is initiated.
8. To share comments and attachments with all group collaborators or workflow participants for a task, select **Share attachments and comments**. This information typically displays in the footer region of Oracle BPM Worklist.

27.3.6.2.2 Creating a Parallel Task Participant List

Users assigned to the list of participants can act upon tasks. You can create several types of lists:

- Value-based name and expression lists
- Value-based management chain lists
- Rule-based names and expression lists
- Rule-based management chain lists
- Rule-based links

For information about creating these lists of participants, see section [Section 27.3.6.1.1, "Creating a Single Task Participant List."](#)

27.3.6.2.3 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 **Deadlines** section (known as the routing slip level) of the Human Task Editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.

To specify a time limit for acting on a task:

1. In the **Advanced** section of the Edit Participant Type dialog for the parallel type, click the **Advanced** icon to expand the section shown in [Figure 27–37](#).
2. Select **Limit allocated duration to**.
3. Specify the amount of time.

For more information about setting the global escalation and renewal policies in the **Deadlines** section of the Human Task Editor, see [Section 27.3.9, "How to Escalate, Renew, or End the Task."](#)

27.3.6.2.4 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.

To invite additional participants to a task:

1. In the **Advanced** section of the Edit Participant Type dialog for the parallel type, click the **Advanced** icon to expand the section (if not expanded).
2. Select **Allow this participant to invite other participants**.

27.3.6.2.5 Bypassing a Task Participant

You can bypass a task participant (user, group, or application role) if a specific condition is satisfied. For example, if a user submits a business trip expense report that is under a specific amount, no approval is required by their manager.

To bypass a task participant:

1. In the Edit Participant Type dialog for the parallel type, select the **Specify skip rule** checkbox.

This action displays an icon for accessing the Expression Builder dialog for building a condition. The expression must evaluate to a boolean value.

For information about a valid XPath expression for skipping a participant, see [Section 27.3.6.1.4, "Bypassing a Task Participant."](#)

27.3.6.3 Configuring the Serial Participant Type

[Figure 27–39](#) displays the Serial dialog.

This participant type enables you to create a list of sequential participants for a workflow. For example, if you want a document to be reviewed by John, Mary, and Scott in sequence, use this participant type. For the serial participant type, they can be any list of users or groups.

Figure 27-39 Edit Participant Type — Serial Type

Type:

Serial

Label:

Stage1.Participant1

e.g., Approval Manager

Participant List

Build a list of participants using:

Names and expressions

Specify attributes using:

Value-based

Rule-based

Participant Names

Identification Type

Data Type

Value

Builds a serial list of the specified participants

Advanced

Limit allocated duration to:

Day

0

Hour

0

Minutes

0

Over all task duration is currently set to {0} days, {1} hours, and {2} minutes.

Allow this participant to invite other participants

Specify skip rule

To configure the serial participant type:

- 1. In the **Label** field, enter a recognizable label for this participant. This label must be unique among all the participants in the task definition (for example, Approval Manager, Primary Reviewers, and so on).

Instructions for configuring the following subsections of the Edit Participant Type dialog for the serial participant type are listed in [Table 27-9](#).

Table 27-9 Edit Participant Type — Serial Type

For This Subsection...	See...
Participant List	Section 27.3.6.3.1, "Creating a Serial Task Participant List"
Limit allocated duration to (under the Advanced section)	Section 27.3.6.3.2, "Specifying a Time Limit for Acting on a Task"
Allow this participant to invite other participants (under the Advanced section)	Section 27.3.6.3.3, "Inviting Additional Participants to a Task"
Specify skip rule (under the Advanced section)	Section 27.3.6.3.4, "Bypassing a Task Participant"

27-36 Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite

27.3.6.3.1 Creating a Serial Task Participant List

Users assigned to the list of participants can act upon tasks. You can create several types of lists:

- Value-based name and expression lists
- Value-based management chain lists
- Rule-based names and expression lists
- Rule-based management chain lists
- Rule-based lists

See section [Section 27.3.6.1.1, "Creating a Single Task Participant List"](#) for instructions on creating these lists of participants.

27.3.6.3.2 Specifying a Time Limit for Acting on a Task

You can specify the amount of time a user, group, or application role receives to act on a task. If the user, group, or role does not act in the time specified, the global escalation and renewal policies that you set in the **Deadlines** section (known as the routing slip level) of the Human Task Editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.

To specify a time limit for acting on a task:

1. In the **Advanced** section of the Edit Participant Type dialog for the serial type, click the **Advanced** icon to expand the section shown in [Figure 27–39](#).
2. Click **Limit allocated duration to**.
3. Specify the amount of time.

For more information about setting the global escalation and renewal policies in the **Deadlines** section of the Human Task Editor, see [Section 27.3.9, "How to Escalate, Renew, or End the Task."](#)

27.3.6.3.3 Inviting Additional Participants to a Task

You can allow a task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow. For example, assume the approval workflow goes from James Cooper to John Steinbeck. If this option is checked, James Cooper can decide to first route it to Irving Stone before it goes to John Steinbeck.

To invite additional participants to a task:

1. In the **Advanced** section of the Edit Participant Type dialog for the serial type, click the **Advanced** icon to expand the section (if not already expanded).
2. Select **Allow this participant to invite other participants**.

Note: For the serial participant type, additional participants can be invited as follows:

- Globally specifying that the ad hoc participants can be invited at anytime. In this case, even in a sequential workflow, approvers can invite other participants at any level in the sequential workflow.
 - Specifying that an ad hoc invitation of other participants can be done only in specific points in the workflow. In this case, other ad hoc participants are invited only when a serial in complete.
-

27.3.6.3.4 Bypassing a Task Participant

You can bypass a task participant (user, group, or application role) if a specific condition is satisfied. For example, if a user submits a business trip expense report that is under a specific amount, no approval is required by their manager.

To bypass a task participant:

1. In the **Advanced** section of the Edit Participant Type dialog for the serial type, select the **Specify skip rule** checkbox.

This action displays an icon for accessing the Expression Builder dialog for building a condition. The expression must evaluate to a boolean value.

For more information about a valid XPath expression for skipping a participant, see [Section 27.3.6.1.4, "Bypassing a Task Participant."](#)

27.3.6.4 Configuring the FYI Participant Type

[Figure 27–40](#) displays the Edit Participant Type dialog for the FYI type.

This participant type is used when a task is sent to a user, but the business process does not wait for a user response; it just continues. FYIs cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.

For example, a magazine subscription is due for renewal. If the user does not cancel the current subscription before the expiration date, the subscription is renewed. This user is reminded weekly until the request expires or the user acts on it.

Figure 27–40 Edit Participant Type — FYI Type

Type: FYI Label: Stage1.Participant1
e.g., Approval Manager

Participant List

Build a list of participants using: Names and expressions

Specify attributes using: ☒ Value-based ☐ Rule-based

Participant Names		
Identification Type	Data Type	Value

To configure the FYI participant type:

1. In the **Label** field, enter a recognizable label for this participant. This label must be unique among all the participants in the task definition (for example, Approval Manager, Primary Reviewers, and so on).

27.3.6.4.1 Creating an FYI Task Participant List

Users assigned to the list of participants can act upon tasks. You can create several types of lists:

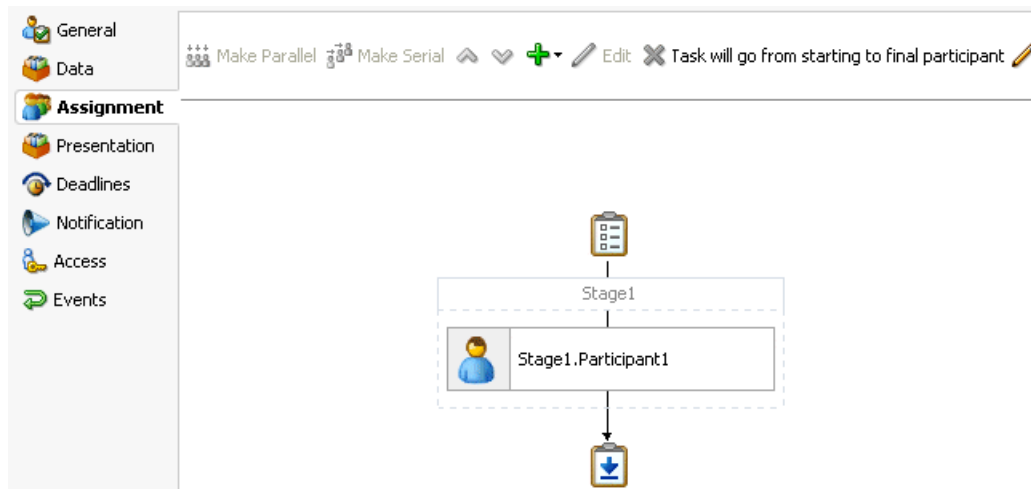
- Value-based name and expression lists
- Value-based management chain lists
- Rule-based names and expression lists
- Rule-based management chain lists
- Rule-based lists

See section [Section 27.3.6.1.1, "Creating a Single Task Participant List"](#) for instructions on creating these lists of participants.

27.3.7 How to Select a Routing Policy

After you configure a participant type and are returned to the Human Task Editor, click the **Task will go from starting to final participant** icon, as shown in [Figure 27–41](#).

Figure 27–41 Human Task Editor — Assignment Section



This displays the Configure Assignment dialog shown in [Figure 27–42](#) for specifying a method for routing your task through the workflow.

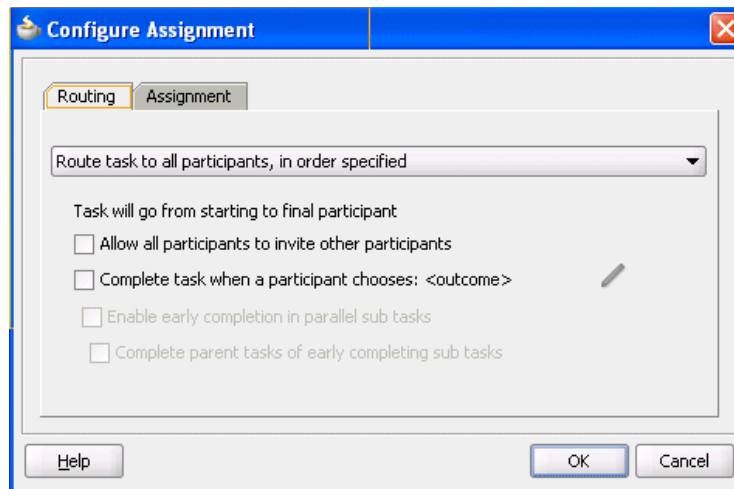
Figure 27–42 Configure Assignment

Table 27–10 describes the routing policy methods provided.

Table 27–10 Routing Policy Method

Routing Policy Selection	Use This Policy In Environments Where...	Section
Route task to all participants, in order specified This selection enables you to specify the following suboptions:	A task must be routed to each of the participants in the order in which they appear. This is predetermined, default routing. For example, in a hiring process, if three users interview and provide review feedback, then the task is sent to the human resources department.	Section 27.3.7.1, "Routing Tasks to All Participants in the Specified Order"
■ Allow all participants to invite other participants	A participant can select users or groups as the next assignee (ad hoc) when approving the task.	Section 27.3.7.1.1, "Allowing All Participants to Invite Other Participants"
■ Complete task when a participant chooses: <outcome>	A participant in a task can accept or reject it, thus ending the workflow without the task being sent to any other participant. For example, a manager rejects a purchase order, meaning that purchase order is not sent to their manager for review.	Section 27.3.7.1.2, "Stopping Routing of a Task to Further Participants"
■ Enable early completion in parallel subtasks	Note: This option is for environments in which you have multiple stages and participants working in parallel. Participants perform subtasks in parallel, and one group's rejection or approval of a subtask does not cause the other group's subtask to also be rejected or approved.	Section 27.3.7.1.3, "Enabling Early Completion in Parallel Subtasks"
■ Complete parent tasks of early completing subtasks	Note: This option is for environments in which you have multiple stages and participants working in parallel. Participants perform subtasks in parallel, and one group's rejection or approval of a subtask causes the other group's subtask to also be rejected or approved.	Section 27.3.7.1.4, "Completing Parent Subtasks of Early Completing Subtasks"

Table 27–10 (Cont.) Routing Policy Method

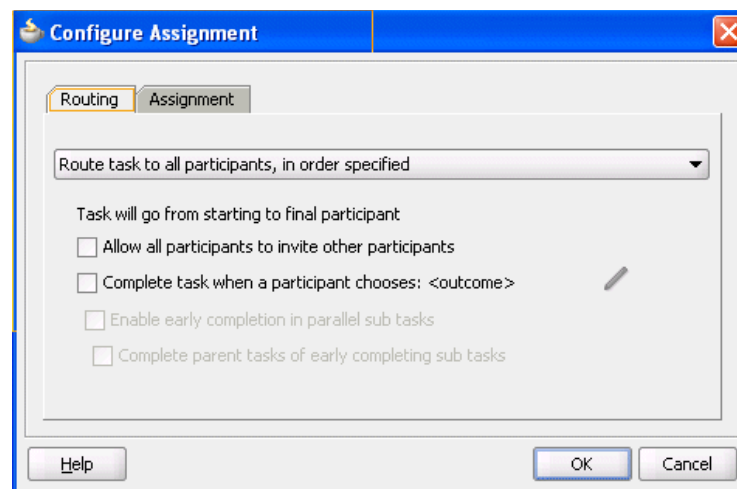
Routing Policy Selection	Use This Policy In Environments Where...	Section
Use Advanced Rules	The participants to whom the task is routed are determined by the business rule logic that you model. For example, a loan application task is designed to go through a loan agent, their manager, and then the senior manager. If the loan agent approves the loan, but their manager rejects it, the task is returned to the loan agent.	Section 27.3.7.2, "Specifying Advanced Task Routing Using Business Rules"
Use External Routing	The participants in a task are dynamically determined. For example, a company's rules may require the task participants to be determined and then retrieved from a back-end database during runtime.	Section 27.3.7.3, "Using External Routing"
Assignment tab	A participant is assigned a failed task for the purposes of recovery.	Section 27.3.7.4, "Configuring the Error Assignee"

27.3.7.1 Routing Tasks to All Participants in the Specified Order

You can select to have a task reviewed by all selected participants. This is known as default routing because the task is routed to each of the participants in the order in which they appear. This type of routing differs from state machine-based routing.

To route tasks to all participants in the specified order:

1. In the **Assignment** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Route task to all participants, in order specified** from the list shown in [Figure 27–43](#).

Figure 27–43 Route a Task to All Participants

See the following tasks to define a routing policy:

- Allowing all participants to invite other participants
- Completing a task when a participant chooses
- Enabling early completion in parallel subtasks
- Completing parent subtasks of early completing subtasks

27.3.7.1.1 Allowing All Participants to Invite Other Participants This checkbox is the equivalent of the ad hoc workflow pattern of pre-10.1.3 Oracle BPEL Process Manager releases. This applies when there is at least one participant. In this case, each user selects users or groups as the next assignee when approving the task.

To allow all participants to invite other participants:

1. In the **Assignment** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Route task to all participants, in order specified**.
3. Select the **Allow all participants to invite other participants** checkbox for this task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow.

27.3.7.1.2 Stopping Routing of a Task to Further Participants You can specify conditions under which to complete a task early, regardless of the other participants in the workflow.

For example, assume an expense report goes to the manager, and then the director. If the first participant (manager) rejects it, you can end the workflow without sending it to the next participant (director).

To abruptly complete a condition:

1. In the **Assignment** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Route task to all participants, in order specified** from the list.
3. Select the **Complete task when a participant chooses: <outcome>** checkbox.

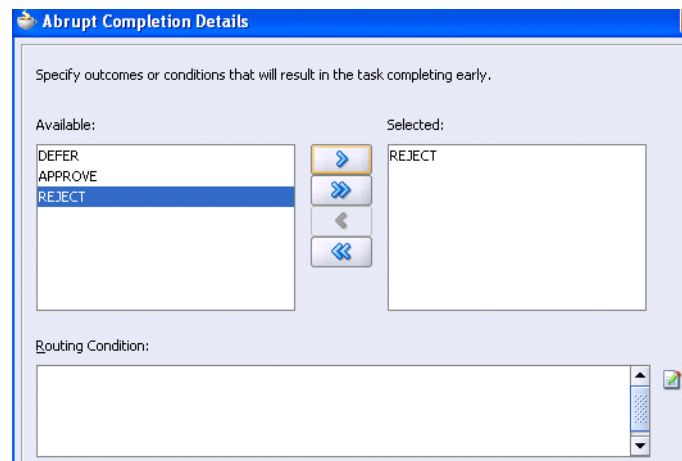
The Abrupt Completion Details dialog appears.

There are two methods for specifying the abrupt completion of a task:

- Outcomes
- XPath expression routing condition

If outcomes are specified, any time the selected task outcome occurs, the task completes. If both outcome and routing condition are specified, the workflow service performs a logical OR operation on the two.

4. Select appropriate outcomes and click the > button, as shown in [Figure 27–44](#). To select all, click the >> button.

Figure 27–44 Abrupt Completion Details

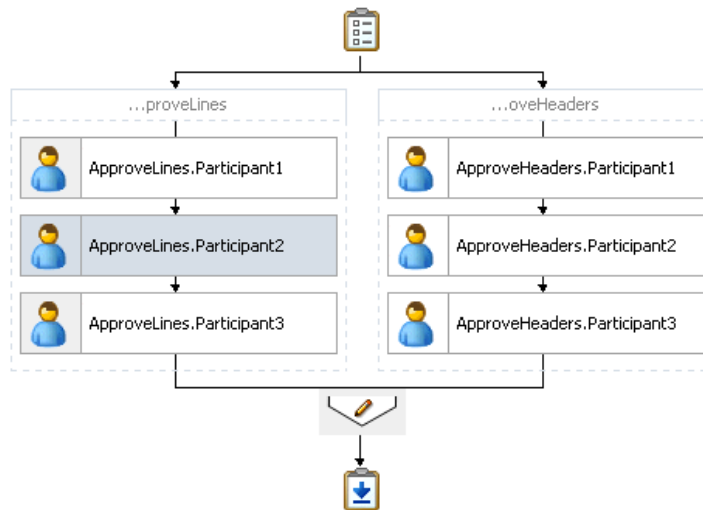
5. To the right of the **Routing Condition** field, click the icon to display the Expression Builder dialog for dynamically creating a condition under which to complete this task early. For example, if a user submits a business trip expense report that is under a specific amount, no approval is required by their manager.
Note that an early completion XPath expression is not evaluated until at least one user has acted upon the task.
6. To enable early completion, click **Enable early completion in parallel with subtasks**. For more information, see [Section 27.3.7.1.3, "Enabling Early Completion in Parallel Subtasks."](#)
7. To enable early completion of parent tasks, click **Complete parent tasks of early completing subtasks**. For more information, see [Section 27.3.7.1.4, "Completing Parent Subtasks of Early Completing Subtasks."](#)
8. Click **OK** to return to the Human Task Editor.

You can click the icon to the right of the **Complete task when a participant chooses: <outcome>** checkbox to edit this information.

27.3.7.1.3 Enabling Early Completion in Parallel Subtasks You can use this option in the following environments:

- Multiple stages and groups of participants perform subtasks in parallel.
- A participant in one group approves or rejects a subtask, which causes the other participants in that same group to stop acting upon the task. However, this does not cause the other parallel group to stop acting upon subtasks. That group continues taking actions on tasks.

For example, assume there are two parallel subgroups, each in separate stages. One group acts upon lines of a purchase order. The other group acts upon headers of the same purchase order. If participant **ApproveLines.Participant2** of the first group rejects a line, all other task participants in the first group stop acting upon tasks. However, the second parallel group continues to act upon headers in the purchase order. In this scenario, the entire task does not complete early. [Figure 27–45](#) provides details.

Figure 27–45 Early Completion of Parallel Subtasks

27.3.7.1.4 Completing Parent Subtasks of Early Completing Subtasks You can use this option in the following environments:

- Multiple stages and groups of participants perform subtasks in parallel.
- A participant in one group approves or rejects a subtask, which causes the other participants in that same group to stop acting upon the task. This also causes the other parallel group to stop acting upon subtasks.

For example, assume there are two parallel subgroups, each in separate stages, as shown in [Figure 27–45](#). One group acts upon lines of a purchase order. The other group acts upon headers of the same purchase order. If participant **ApproveLines.Participant2** of the first group rejects a line, all other task participants in the first group stop acting upon tasks. In addition, the second parallel group stops acting upon headers in the purchase order. In this scenario, the entire task completes early.

27.3.7.2 Specifying Advanced Task Routing Using Business Rules

Use advanced routing rules to create complex workflow routing scenarios. The participant types (single, parallel, serial, and FYI) are used to create a linear flow from one set of users to another with basic conditions such as abrupt termination, skipping assignees, and so on. However, there is often a need to perform more complex back and forth routing between multiple individuals in a workflow. One option is to use the BPEL process as the orchestrator of these tasks. Another option is to specify it declaratively using business rules. This section describes how you can model such complex interactions by using business rules with the Human Task Editor.

27.3.7.2.1 Introduction to Advanced Task Routing Using Business Rules You can define state machine routing rules using Oracle Business Rules. This action enables you to create Oracle Business Rules that are evaluated:

- After a routing slip task participant sets the outcome of the task
- Before the task is assigned to the next routing slip participant

This action enables you to override the standard task routing slip method described in [Section 27.3.7.1, "Routing Tasks to All Participants in the Specified Order"](#) and build complex routing behavior into tasks.

Using Oracle Business Rules, you define a set of rules (called a ruleset) that relies on business objects, called facts, to determine which action to take.

27.3.7.2.2 Facts A fact is an object with certain business data. Each time a routing slip assignee sets the outcome of a task, instead of automatically routing the task to the next assignee, the task service performs the following steps:

- Asserts facts into the decision service
- Executes the advanced routing ruleset

Rules can test values in the asserted facts and specify the routing behavior by setting values in a TaskAction fact type.

[Table 27–11](#) describes the fact types asserted by the task service.

Table 27–11 Fact Types Asserted By the Task Service

Fact Type	Description
Task	This fact contains the current state of the workflow task instance. All task attributes can be tested against it. The task fact also contains the current task payload. This fact enables you to construct tests against payload values and task attribute values.
PreviousOutcome	This fact describes the previous task outcome and the assignee who set the outcome. The previous outcome fact contains the following attributes: ■ <code>actualParticipant</code>: The name of the participant who set the task outcome (for example, <code>jstein</code>) ■ <code>logicalParticipant</code>: The logical name (or label) for the routing slip participant responsible for setting the task outcome (for example, <code>assignee1</code>) ■ <code>outcome</code>: The outcome that was set (for example, <code>approve</code> or <code>reject</code>) ■ <code>level</code>: If the previous participant was part of a management chain, then this attribute records their level in the chain, where 1 is the first level in the chain. For other participant types, the value is -1. ■ <code>totalNumberOfApprovals</code>: The total number of users that have now set the outcome of the task.
TaskAction	This fact is not intended for writing rule tests against it. Instead, it is updated by the ruleset, and returned to the task service to indicate how the task should be routed. Rules should not directly update the TaskAction fact. Instead, they should call one of the RL functions described in Section 27.3.7.2.3, "Action Types." These functions handle updating the TaskAction fact with the appropriate values.

Some fact types can only be used in workflow routing rules, while others can only be used in workflow participant rules. [Table 27–12](#) describes where you can use each type.

Table 27–12 Use of Fact Types

Fact Type	Can Use in Routing Rules?	Can Use in Participant Rules?
Task	Yes	Yes
PreviousOutcome	Yes	No
TaskAction	Yes	No
Lists	No	Yes
RoutingSlipObjectFactory	No	Yes

Table 27–12 (Cont.) Use of Fact Types

Fact Type	Can Use in Routing Rules?	Can Use in Participant Rules?
ResourceListType	No	Yes
ManagementChainListType	No	Yes
ResourceType	No	Yes
ParameterType	No	Yes
AutoActionType	No	Yes
ResponseType	No	Yes

27.3.7.2.3 Action Types To instruct the task service on how to route the task, rules can specify one of many task actions. This is done by updating the `TaskAction` fact asserted into the rule session. However, rules should not directly update the `TaskAction` fact. Instead, rules should call one of the action RL functions, passing the `TaskAction` fact as a parameter. These functions handle the actual updates to the fact. For example, to specify an action of go forward, you must add a `call GO_FORWARD(TaskAction)` to the action part of the rule.

Each time a state machine routing rule is evaluated, the rule takes one of the actions shown in [Table 27–13](#):

Table 27–13 Business Rule Actions

Action	Description	Parameters
GO_FORWARD	Goes to the next participant in the routing slip (default behavior).	None
PUSHBACK	Goes back to the previous participant in the routing slip (the participant before the one that just set the task outcome).	None
GOTO	Goes to a specific participant in the routing slip.	participant' A string that identifies the label of the participant (for example, <code>Approver1</code>) to which to route the task.
COMPLETE	Finishes routing and completes the task. The task is marked as completed, and no further routing is required.	None
ESCALATE	Escalates and reassigns the task according to the task escalation policy (usually to the manager of the current assignee).	None

27.3.7.2.4 Sample Ruleset This section describes how to use rules to implement custom routing behavior with a simple example. A human workflow task is created for managing approvals of expense requests. The outcomes for the task are approve and reject. The task definition includes an `ExpenseRequest` payload element. One of the fields of `ExpenseRequest` is the total amount of the expense request. The routing slip for the task consists of three single participants (`assignee1`, `assignee2`, and `assignee3`).

By default, the task gets routed to each of the assignees, with each assignee choosing to approve or reject the task.

Instead of this behavior, the necessary routing behavior is as follows:

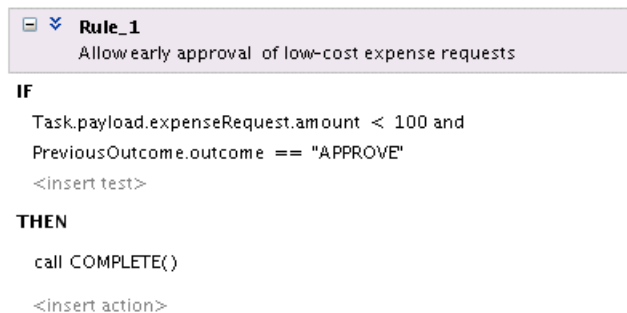
- If the total amount of the expense request is less than \$100, approval is only required from one of the participants. Otherwise, it must be approved by all three.
- If an expense request is rejected by any of the participants, it must be returned to the previous participant for re-evaluation. If it is rejected by the first participant, the expense request is rejected and marked as completed.

This behavior is implemented using the following rules. Note that when a rule dictionary is generated for advanced routing rules, it is created with a template rule that implements the default `GO_FORWARD` behavior. You can edit this rule, and make copies of the template rule by right-clicking and selecting **Copy Rule** in the Oracle Business Rules Designer.

If the amount is greater than \$100 and the previous assignee approved the task, it is not necessary to provide a rule for routing a task to each of the assignees in turn. This is the default behavior that is reverted to if none of the rules in the ruleset are triggered:

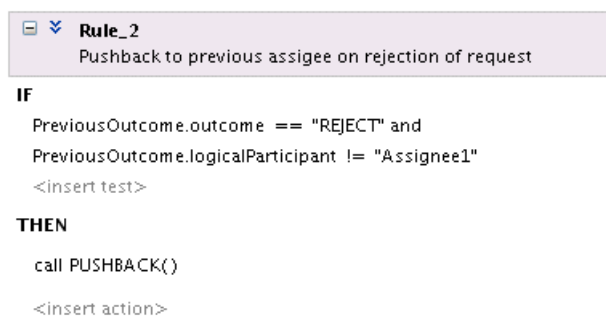
- Early approval rule (Figure 27–46):

Figure 27–46 Early Approval Rule

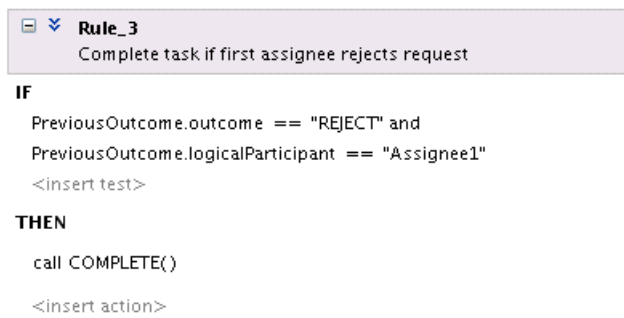


- Push back on the rejected rule (Figure 27–47):

Figure 27–47 Push Back On The Rejected Rule



- Complete the Assignee1 rejected rule (Figure 27–48):

Figure 27-48 Completion of the Assignee1 Rejected Rule

For information about iterative design, see the `workflow-106-IterativeDesign` sample available at the Oracle Technology Network:

<https://soasamples.samplecode.oracle.com>

27.3.7.2.5 Linked Dictionary Support For human workflow, business rule artifacts are now stored in two rules dictionaries. This is useful for scenarios in which you must customize your applications. For example, you create and ship version 1 of an application to a customer. The customer then customizes the rulesets in the application with Oracle SOA Composer. Those customizations are now stored in a different rules dictionary than the base rules dictionary. The rules dictionary that stores the customized rulesets links with the rules in the base dictionary. When you later ship version 2 of the application, the base rule dictionary may contain additional changes introduced in the product. The ruleset customization changes previously performed by the customer are preserved and available with the new changes in the base dictionary. When an existing application containing a task using rules is opened, if the rules are in the old format using one dictionary, they are automatically upgraded and divided into two rules dictionaries:

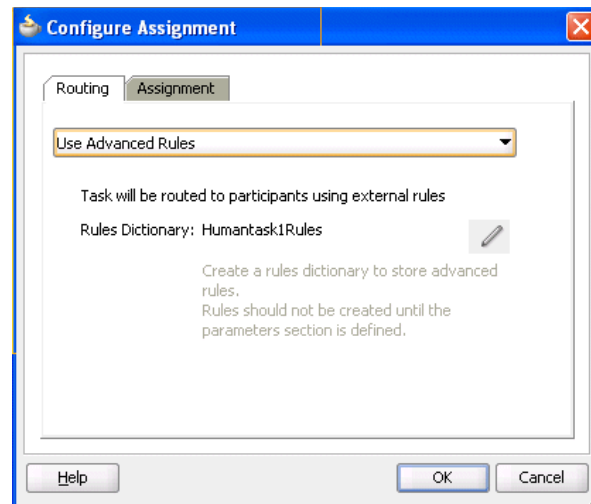
- Base dictionary
- Custom dictionary

For more information about customizations, see [Chapter 43, "Customizing SOA Composite Applications."](#)

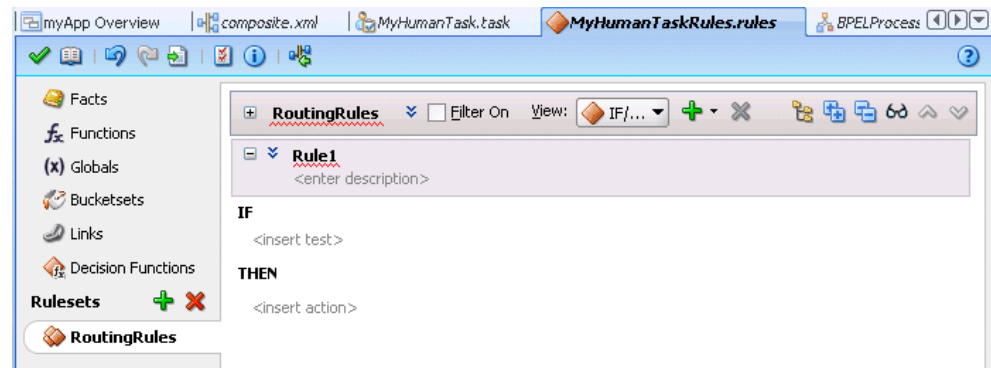
27.3.7.2.6 Creating Advanced Routing Rules

To create advanced routing rules:

1. In the **Assignment** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Use Advanced Rules** from the list.
3. To the right of **Rules Dictionary**, click the **Edit** icon, as shown in [Figure 27-49](#).

Figure 27–49 Creating a Rules Dictionary

This starts the Oracle Business Rules Designer with a preseeded repository containing all necessary fact definitions, as shown in [Figure 27–50](#). A decision service component is created for the dictionary, and is associated with the task service component.

Figure 27–50 Human Task Rule Dictionary

4. Define state machine routing rules for your task using Oracle Business Rules.

This automatically creates a fully-wired decision service in the human task and the associated rule repository and data model.

For more information about business rules, see the following documentation:

- [Section 27.3.7.2.4, "Sample Ruleset"](#) for an example human task ruleset
- *Oracle Fusion Middleware User's Guide for Oracle Business Rules*
- *Oracle Fusion Middleware Language Reference Guide for Oracle Business Rules*

27.3.7.3 Using External Routing

You configure an external routing service that dynamically determines the participants in the workflow. If this routing policy is specified, all other participant types are ignored. It is assumed that the external routing service provides a list of participant types (single approver, serial approver, parallel approver, and so on) at runtime to determine the routing of the task.

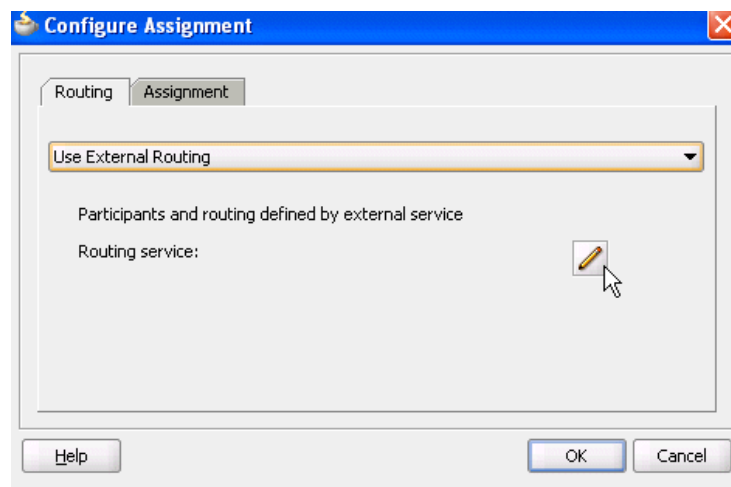
Use this option if you do not want to use any of the routing rules to determine task assignees. In this case, all the logic of task assignment is delegated to the external routing service.

Note: If you select **Use External Routing** in the Configure Assignment dialog, specify a Java class, and click **OK** to exit, the next time you open this dialog, the other two selections (**Route task to all participants, in order specified** and **Use Advanced Rules**) no longer appear in the dropdown list. To access all three selections again, you must delete the entire assignment.

To use external routing

1. In the **Assignment** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Use External Routing** from the list.
3. Click the **Edit** icon, as shown in [Figure 27-51](#).

Figure 27-51 Selection of *Use External Routing*



The External Routing dialog appears, as shown in [Figure 27-52](#).

Figure 27–52 Use External Routing Dialog

4. In the **Class Name** field, enter the fully qualified class file name (for example, the `org.mycompany.tasks.RoutingService` class name). This class must implement the following interface:

```
oracle.bpel.services.workflow.task.IAssignmentService
```

5. Add name and pair value parameters by name or XPath expression that can be passed to the external service, as shown in [Table 27–14](#).

Table 27–14 External Routing

Field	Description
By Name	Enter a name in the Name field and a value in the Value field.
By Expression	Enter a name and dynamically enter a value by clicking the icon to the right of the field to display the Expression Builder dialog.

6. Click the **Add** icon to add additional name and pair value parameters.

27.3.7.4 Configuring the Error Assignee

Tasks can error for reasons such as incorrect assignments. When such errors occur, the task is assigned to the error assignee, who can perform corrective actions. Recoverable errors are as follows:

- Invalid user and group for all participants
- Invalid XPath expressions that are related to assignees and expiration duration
- Escalation on expiration errors
- Evaluating escalation policy
- Evaluating renewal policy
- Computing a management chain
- Evaluating dynamic assignment rules. The task is not currently in error, but is still left as assigned to the current user and is therefore recoverable.

- Dynamic assignment cyclic assignment (for example, user A > user B > user A). The task is not currently in error, but is still left as assigned to the last user in the chain and is therefore recoverable.

The following errors are not recoverable. In these cases, the task is moved to the terminating state `ERRORED`.

- Invalid task metadata
- Unable to read task metadata
- Invalid GOTO participant from state machine rules
- Assignment service not found
- Any errors from assignment service
- Evaluating custom escalate functions
- Invalid XPath and values for parallel default outcome and percentage values

During modeling of workflow tasks, you can specify error assignees for the workflow. If error assignees are specified, they are evaluated and the task is assigned to them. If no error assignee is specified at runtime, an administration user is discovered and is assigned the alerted task. The error assignee can perform one of the following actions:

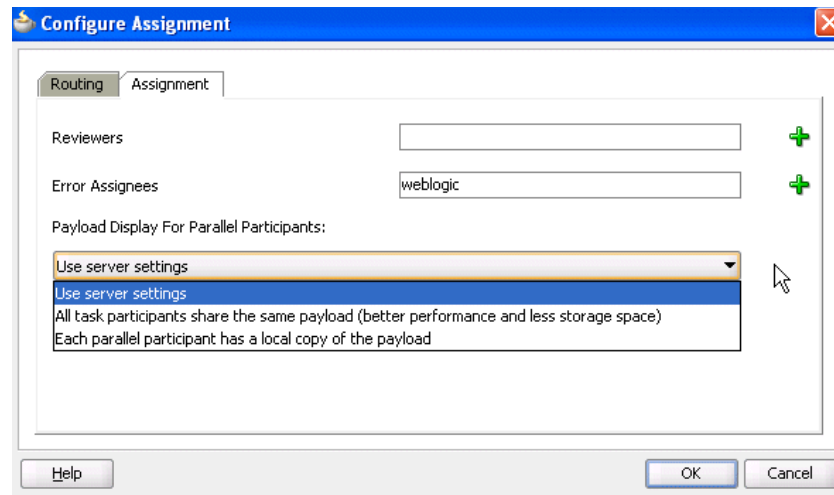
- Ad hoc route
Route the task to the actual users assigned to the task. Ad hoc routing allows the task to be routed to users in sequence, parallel, and so on.
- Reassign
Reassign the task to the actual users assigned to this task
- Error task
Indicate that this task cannot be rectified.

If there are any errors in evaluating the error assignees, the task is marked as being in error.

This dialog enables you to specify the users or groups to whom the task is assigned if an error in assignment has occurred.

To configure the error assignee:

1. In the **Assignment** section, click the icon to the right of **Task will go from starting to final participant**.
2. Click the **Assignment** tab.
3. Click the **Add** icon to assign reviewers or error assignees, as shown in [Figure 27-53](#).

Figure 27–53 Error Assignment Details

4. Click the **Add** icon and select a user, group, or application role to participate in this task.

The **Identification Type** column of the **Starting Participant** table displays your selection of user, group, or application role.

5. See Step 4 through 6 of [Section 27.3.6.1.1, "Creating a Single Task Participant List"](#) for instructions on selecting a user, group, or application role.
6. If you are using parallel participant types, you can specify where to store the subtask payload with the following options.

- **Use server settings**

The **SharePayloadAcrossAllParallelApprovers** System MBean Browser boolean property in Oracle Enterprise Manager Fusion Middleware Control determines whether to share the payload of subtasks in the root task. By default, this property is set to **true**. If set to **true**, the **All task participants share the same payload (better performance and less storage space)** option is used. If this property is set to **false**, the **Each parallel participant has a local copy of the payload** option is used. To change this property, perform the following steps:

- a. Right-click **soa-infra** and select **Administration > System MBean Browser**.
- b. Expand **Application Defined MBeans > oracle.as.soainfra.config > Server: server_name > WorkflowConfig > human-workflow**.
- c. Click **SharePayloadAcrossAllParallelApprovers**.
- d. Change this property in the list, and click **Apply**.

- **All task participants share the same payload (better performance and less storage space)**

The payload for the subtasks is stored in their root task. This means that the payload of the root task is shared across all its subtasks. Internally, this option provides better performance and storage space consumption. Less storage space is consumed because the payload of the root task is shared across all its subtasks.

- **Each parallel participant has a local copy of the payload**

Each subtask has its own copy of the payload. Internally, this option provides lesser performance and storage space consumption because more storage space is consumed.

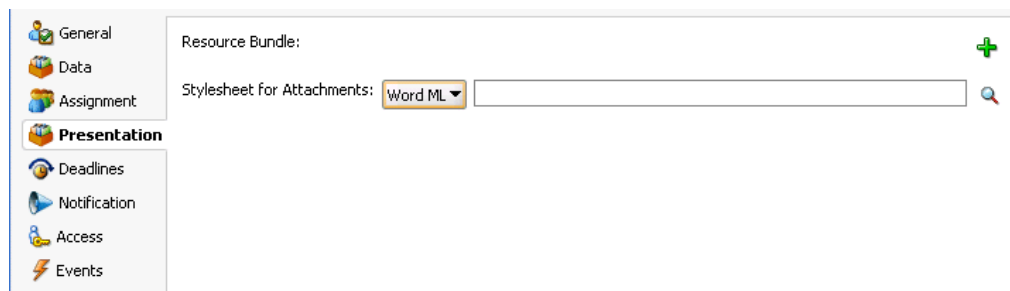
7. Click **OK**.

For more information about users, groups, or application roles, see [Section 26.2.1.1.3, "Participant Assignment."](#)

27.3.8 How to Specify Multilingual Settings and Style Sheets

The **Presentation** section shown in [Figure 27–54](#) enables you to specify resource bundles for displaying task details in different languages in Oracle BPM Worklist and WordML and custom style sheets for attachments.

Figure 27–54 Presentation Section



27.3.8.1 Specifying WordML and Other Style Sheets for Attachments

To specify WordML style sheets for attachments:

- In the **Stylesheet for Attachments** list of the **Presentation** section, select one of the following options:
 - Word ML:** This option dynamically creates Microsoft Word documents for sending as email attachments using a WordML XSLT style sheet. The XSLT style sheet is applied on the task document.
 - Other:** This option creates email attachments using an XSLT style sheet. The XSLT style sheet is applied on the task document.
- Click the **Search** icon to select the style sheet as an attachment.

27.3.8.2 Specifying Multilingual Settings

You can specify resource bundles for displaying task details in different languages in Oracle BPM Worklist. Resource bundles are supported for the following task details:

- Displaying the value for task outcomes in plain text or with the message (key) format.
- Making email notification messages available in different languages. At runtime, you specify the `hwf:getTaskResourceBundleString(taskId, key, locale?)` XPath extension function to obtain the internationalized string from the specified resource bundle. The locale of the notification recipient can be retrieved with the function `hwf:getNotificationProperty(propertyName)`.

Resource bundles can also simply be property files. For example, a resource bundle that configures a display name for task outcomes can look as follows:

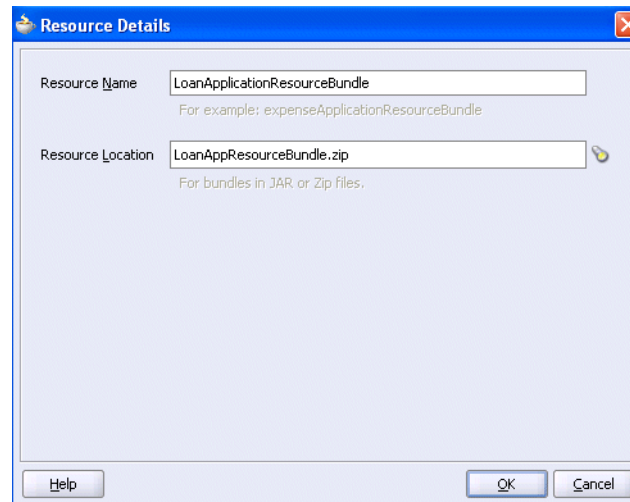
- APPROVE=Approve
- REJECT=Reject

To specify multilingual settings:

1. In the **Presentation** section, click the **Add** icon across from **Resource Bundle**.

The Resource Details dialog shown in [Figure 27–55](#) appears.

Figure 27–55 Resource Details Dialog



2. In the **Resource Name** field, enter the name of the resource used in the resource bundle. This should be a `.properties`-based resource bundle file.
3. In the **Resource Location** field, click the **Search** icon to select the JAR or ZIP resource bundle file to use. The resource bundle is part of your system archive (SAR) file.

If the resource bundle is outside of the composite project, you are prompted to place a local copy in `SCA-INF/lib`.

If the resource bundle file is not in the composite class loader (directly under `SCA-INF/classes` or in a JAR file in `SCA-INF/lib`), you must specify its location. For example, if the resource bundle is accessible from a location outside of the composite class loader (for example, an HTTP location such as `http://host:port/bundleApp/taskBundles.jar`), then this location must be specified in this field.

4. Click **OK** to return to the Human Task Editor.

For more information, see [Section 31.2.6, "How to Configure Notification Messages in Different Languages."](#)

27.3.9 How to Escalate, Renew, or End the Task

[Figure 27–56](#) shows the **Deadlines** section of the Human Task Editor.

You can specify the expiration duration of a task in this global policy section (also known as the routing slip level). If the 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**

checkbox), then those settings take precedence over settings specified in the **Deadlines** section (routing slip level).

You can also specify that a task be escalated to a user's manager after a specified time period. For more information, see [Section 27.3.6.1.2, "Specifying a Time Limit for Acting on a Task."](#)

Figure 27–56 Human Task Editor — Deadlines Section

Task Duration Settings: **Escalate after**

Fixed Duration: Day 0 Hour 0 Minutes 0

Maximum Escalation Levels: 0

Highest Approver Title:

Custom Escalation Java Class:

☒ Action Requested Before : By Duration Day 0 Hour 0 Minutes 0

27.3.9.1 Introduction to Escalation and Expiration Policy

This section provides an overview of how specifying the expiration duration at this level makes this setting the expiration duration of the task across all the participants.

For example, participant **LoanAgentGroup** and participant **Supervisor** have three days to act on the task between them, as shown in [Figure 27–57](#):

Figure 27–57 Expire After Policy

Task Duration Settings: **Expire after**

Fixed Duration: Day 3 Hour 0 Minutes 0

Custom Escalation Java Class:

☐ Action Requested Before :

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 level of the participants, then for that participant, the participant expiration duration is used. However, the global expiration duration is still used for the participants that do not have participant level expiration duration. The global expiration duration is always decremented by the time elapsed in the task.

The policy for interpreting the participant level expiration for the participants is described as follows:

- **Serial**

Each assignment in the management chain gets the same expiration duration as the one specified in the serial. Note that the duration is not for all the assignments resulting from this assignment. If the task expires at any of the assignments in the management chain, the escalation and renewal policy is applied.

- **Parallel:**

- In a parallel workflow, if the parallel participants are specified as a resource, a routing slip is created for each of the resources. The expiration duration of each created routing slip follows these rules:
 - * The expiration duration equals the expiration duration of the parallel participant if it has an expiration duration specified.
 - * The expiration duration that is left on the task if it was specified at the routing slip level.
 - * Otherwise, there is no expiration duration.
- If parallel participants are specified as routing slips, then the expiration duration for the parallel participants is 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.

27.3.9.2 Specifying a Policy to Never Expire

You can specify for a task to never expire.

To specify a policy to never expire:

1. In the dropdown list in the **Deadlines** section, as shown in [Figure 27–56](#), select **Never Expire**.

27.3.9.3 Specifying a Policy to Expire

You can specify for a task to expire. When the task expires, either the escalation policy or the renewal policy at the routing slip level is applied. If neither is specified, the task expires. The expiration policy at the routing slip level is common to all the participants.

To specify for a task to expire:

1. In the dropdown list of the **Deadlines** section, select **Expire after**, as shown in [Figure 27–58](#).
2. Specify the maximum time period for the task to remain open.

The expiration policy for parallel participants is interpreted as follows:

- If parallel participants are specified as resources in parallel elements, there is no expiration policy for each of those participants.
- If parallel participants are specified as routing slips, then the expiration policy for the routing slip applies to the parallel participants.

[Figure 27–58](#) indicates that the task expires in three days.

Figure 27–58 Expire After Policy

Task Duration Settings: Expire after ▾

Fixed Duration ▾ Day 3 ▴ ▾ Hour 0 ▴ ▾ Minutes 0 ▴ ▾

Custom Escalation Java Class:

☐ Action Requested Before :

27.3.9.4 Extending an Expiration Policy Period

You can extend the expiration period when the user does not respond within the allotted time. You do this by specifying the number of times the task can be renewed upon expiration (for example, renew it an additional three times) and the duration of each renewal (for example, three days for each renewal period).

To extend an expiration policy period:

1. In the dropdown list of the **Deadlines** section, select **Renew after**, as shown in [Figure 27–59](#).
2. Specify the maximum number of times to continue renewing this task.

In [Figure 27–59](#), when the task expires, it is renewed at most three times. It does not matter if the task expired at the **LoanAgentGroup** participant or the **Supervisor** participant.

Figure 27–59 Renew After Policy

Task Duration Settings: Renew after ▾

Fixed Duration ▾ Day 0 ▴ ▾ Hour 0 ▴ ▾ Minutes 0 ▴ ▾

Maximum Renewals: 3

Custom Escalation Java Class:

☐ Action Requested Before :

27.3.9.5 Escalating a Task Policy

You can escalate a task if a user does not respond within the allotted time. For example, if you are using the escalation hierarchy configured in your user directory, the task can be escalated to the user's manager. If you are using escalation callbacks, the task is escalated to whoever you have defined. When a task has been escalated the maximum number of times, it stops escalating. An escalated task can remain in a user inbox even after the task has expired.

To escalate a task policy:

1. In the dropdown list of the **Deadlines** section, select **Escalate after**, as shown in [Figure 27–60](#).
2. Specify the following additional values. When both are set, the escalation policy is more restrictive.

- **Maximum Escalation Levels**

Number of management levels to which to escalate the task. This field is required.

- **Highest Approver Title**

The title of the highest approver (for example, self, manager, director, or CEO). These titles are compared against the title of the task assignee in the corresponding user repository. This field is optional.

The escalation policy specifies the number of times the task can be escalated on expiration and the renewal duration. In [Figure 27–60](#), when the task expires, it is escalated at most three times. It does not matter if the task expired at the **LoanAgentGroup** participant or the **Supervisor** participant.

Figure 27–60 Escalate After Policy

Task Duration Settings:

Escalate after ▼

Fixed Duration ▼ Day 0 ↑ ↓ Hour 0 ↑ ↓ Minutes 0 ↑ ↓

Maximum Escalation Levels 3

Highest Approver Title: ▼

Custom Escalation Java Class:

☐ Action Requested Before :

27.3.9.6 Specifying Escalation Rules

This option allows a custom escalation rule to be plugged in for a particular workflow. For example, to assign the task to a current user's department manager on task expiration, you can write a custom task escalation function, register it with the workflow service, and use that function in task definitions.

The default escalation rule is to assign a task to the manager of the current user. To add a new escalation rule, follow these steps.

To specify escalation rules:

1. Implement the following interface:

```
oracle.bpel.services.workflow.assignment.dynamic.IDynamicTaskEscalationFunction
```

This implementation must be available in the class path for the server.

2. Log in to Oracle Enterprise Manager Fusion Middleware Control.
3. Expand the **SOA** folder in the navigator.
4. Right-click **soa-infra**, and select **SOA Administration > Workflow Task Service Properties**.

The Workflow Task Service Properties page appears.

5. Add a new function. For example:

- Function name: DepartmentSupervisor
- Classpath:
oracle.bpel.services.workflow.assignment.dynamic.patterns.
DepartmentSupervisor

- Function parameter name
 - Function parameter value
6. In the **Custom Escalation Java Class** field of the **Deadlines** section, enter the function name as defined in the Workflow Task Service Properties page for the escalation rule.

For more information, see [Section 31.3.3, "Custom Escalation Function."](#)

27.3.9.7 Specifying a Due Date

A due date indicates the date by which the task should be completed. Note that the due date is different from the expiration date. When a task expires it is either marked expired or automatically escalated or renewed based on the escalation policy. The due date is generally a date earlier than the expiration date and an indication to the user that the task is about to expire.

You can enter a due date for a task, as shown in [Figure 27–56](#). A task is considered overdue after it is past the specified due date. This date is in addition to the expiration policy. A due date can be specified irrespective of whether an expiration policy has been specified. The due date enables Oracle BPM Worklist to display a due date, list overdue tasks, highlight overdue tasks in the inbox, and so on. Overdue tasks can be queried using a predicate on the `TaskQueryService.queryTask(...)` API.

To specify a due date:

1. In the **Deadlines** section, select the **Action Requested Before** checkbox.
2. Select **By Duration** to enter a time duration or select **By Expression** to dynamically enter a value as an XPath expression.

Note the following details:

- The due date can be set on both the task (using the Create ToDo Task dialog in Oracle BPM Worklist) and in the `.task` file (using the Human Task Editor). This is to allow to-do tasks without task definitions to set a due date during initiation of the task. A due date that is set in the task (a runtime object) overrides a due date that is set in the `.task` file.
- In the task definition, the due date can only be specified at the global level, and not for each participant.
- If the due date is set on the task, the due date in the `.task` file is ignored.
- If the due date is not set on the task, the due date in the `.task` file is evaluated and set on the task.
- If there is no due date on either the task or in the `.task` file, there is no due date on the task.

Note: You cannot specify business rules for to-do tasks.

For more information, see [Section 29.3.4, "How To Create a ToDo Task."](#)

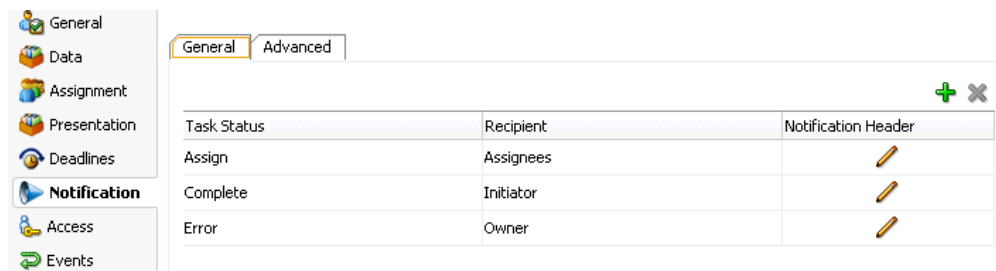
27.3.10 How to Specify Participant Notification Preferences

[Figure 27–61](#) shows the **General** tab of the **Notification** section of the Human Task Editor (when fully expanded).

Notifications indicate when a user or group is assigned a task or informed that the status of the task has changed. Notifications can be sent through email, voice message, instant message, or SMS. Notifications are sent to different types of participants for different actions. Notifications are configured by default with default messages. For example, a notification message is sent to indicate that a task has completed and closed. You can create your own or modify existing configurations.

Note: Embedded LDAP does not support group email addresses. Therefore, when a task is assigned to a group ID, emails are sent to all of its members instead of to the group email address.

Figure 27–61 Human Task Editor — General Tab of Notification Section



To specify participant notification preferences:

1. Click the **Notification** tab (displays as shown in [Figure 27–61](#)).

Instructions for configuring the following subsections of the **General** tab of the **Notification** section are listed in [Table 27–15](#).

Table 27–15 Human Task Editor — General Tab of Notification Section

For This Subsection...	See...
Task Status	Section 27.3.10.1, "Notifying Recipients of Changes to Task Status"
Recipient	
Notification Header	Section 27.3.10.2, "Editing the Notification Message"

For information about the notification service, see [Section 31.2, "Notifications from Human Workflow."](#)

2. In the **Notification** section, click the **Advanced** tab. [Figure 27–62](#) provides details.

Figure 27–62 Notification Section - Advanced Tab

Instructions for configuring the following subsections of the **Advanced** tab of the **Notification** section are listed in [Table 27–16](#).

Table 27–16 Human Task Editor — Advanced Tab of Notification Section

For This Subsection...	See...
Reminders	Section 27.3.10.3, "Setting Up Reminders"
Encoding	Section 27.3.10.4, "Changing the Character Set Encoding"
Make notifications secure (exclude details)	Section 27.3.10.5, "Securing Notifications to Exclude Details"
Show worklist URL in notifications	Section 27.3.10.6, "Showing the Oracle BPM Worklist URL in Notifications"
Make notifications actionable	Section 27.3.10.7, "Making Email Messages Actionable"
Send task attachments with email notifications	Section 27.3.10.8, "Sending Task Attachments with Email Notifications"
Group notification configuration	Section 27.3.10.9, "Sending Email Notifications to Groups and Application Roles"
Notification header attributes	Section 27.3.10.10, "Customizing Notification Headers"

27.3.10.1 Notifying Recipients of Changes to Task Status

Three default status types display in the **Task Status** column: **Assign**, **Complete**, and **Error**. You can select other status types for which to receive notification messages.

To notify recipients of changes to task status:

1. In the **Notification** section, click the **General** tab.
2. In the **Task Status** column, click a type to display the complete list of task types:
 - **Alerted**

When a task is in an alerted state, you can notify recipients. However, none of the notification recipients (assignees, approvers, owner, initiator, or reviewer) can move the task from an alerted state to an error state; they only receive an FYI notification of the alerted state. The owner can reassign, withdraw, delete, or purge the task, or ask the error assignee to move the task to an error state if the error cannot be resolved. Only the error assignee can move a task from an alerted state to an error state.

You configure the error assignee on the **Assignment** tab of the Configure Assignment dialog under the **Task will go from starting to final participant** icon in the **Assignment** section. For more information, see [Section 27.3.7.4, "Configuring the Error Assignee."](#)

- **Assign**

When the task is assigned to users or a group. This captures the following actions:

- Task is assigned to a user
- Task is assigned to a new user in a serial workflow
- Task is renewed
- Task is delegated
- Task is reassigned
- Task is escalated
- Information for a task is submitted

- **Complete**

- **Error**

- **Expire**

- **Request Info**

- **Resume**

- **Suspend**

- **Update**

- Task payload is updated
- Task is updated
- Comments are added
- Attachments are added and updated

- **Update Outcome**

- **Withdraw**

- **All Other Actions**

- Any action not covered in the above task types. This includes acquiring a task.

3. 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.

4. In the **Recipient** column, click an entry to display a list of possible recipients for the notification message:
 - **Assignees**
The users or groups to whom the task is currently assigned.
 - **Initiator**
The user who created the task.
 - **Approvers**
The users who have acted on the task up to this point. This applies in a serial participant type in which multiple users have approved the task and a notification must be sent to all of them.
 - **Owner**
The task owner
 - **Reviewer**
The user who can add comments and attachments to a task.

For more information, see [Section 31.2.5, "How to Configure the Notification Channel Preferences."](#)

27.3.10.2 Editing the Notification Message

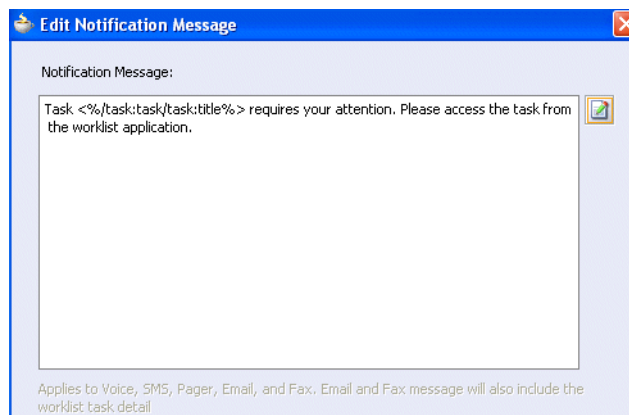
A default notification message is available for delivery to the selected recipient. If you want, you can modify the default message text.

To edit the notification message:

1. In the **Notification** section, click the **General** tab.
2. In the **Notification Header** column, click the **Edit** icon to modify the default notification message.

The Edit Notification Message dialog shown in [Figure 27-63](#) appears.

Figure 27-63 Edit Notification Message Dialog



This message applies to all the supported notification channels: email, voice, instant messaging, and SMS. Email messages can also include the worklist task detail defined in this message. The channel by which the message is delivered is based upon the notification preferences you specify.

3. Modify the message wording as necessary.
4. Click **OK** to return to the Human Task Editor.

For more information about notification preference details, see [Section 31.2, "Notifications from Human Workflow."](#)

27.3.10.3 Setting Up Reminders

You can send task reminders, which can be based on the time the task was assigned to a user or the expiration time of a task. The number of reminders and the interval between the reminders can also be configured.

To set up reminders:

1. In the **Notification** section, click the **Advanced** tab.
2. From the list, select the number of reminders to send.
3. 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.

For more information, see [Section 31.2.12, "How to Send Reminders."](#)

27.3.10.4 Changing the Character Set Encoding

Unicode is a universally-encoded character set that enables information from any language to be stored using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language. You can use the default setting of UTF-8 or you can specify a character set with a Java class.

To change the character set encoding

1. In the **Notification** section, click the **Advanced** tab.
2. From the **Encoding** list, select **Specify by Java Class**.
3. Enter the Java class to use.

27.3.10.5 Securing Notifications to Exclude Details

To secure notifications, make messages actionable, and send attachments:

1. In the **Notification** section, click the **Advanced** tab.
2. Select **Make notifications secure (exclude details)**.

If selected, a default notification message is used. There are no HTML worklist task details, attachments, or actionable links in the email. Only the task number is in the message.

For more information, see [Section 31.2.10, "How to Send Secure Notifications."](#)

27.3.10.6 Showing the Oracle BPM Worklist URL in Notifications

You can configure whether to display the Oracle BPM Worklist URL in email notification messages.

To show the Oracle BPM Worklist URL in notifications:

1. In the **Notification** section, click the **Advanced** tab.

2. Select the **Show worklist URL in notifications** checkbox to display the Oracle BPM Worklist URL in email notification messages. If this checkbox is not selected, the URL is not displayed.

27.3.10.7 Making Email Messages Actionable

To make email messages actionable:

1. In the **Notification** section, click the **Advanced** tab.
2. Select **Make notification actionable**. This action enables you to perform task actions through email.

Note: FYI tasks are not actionable and cannot be acknowledged from email messages.

For more information about additional configuration details, see [Section 31.2.7, "How to Send Actionable Messages."](#)

For more information about configuring outbound and inbound emails, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

27.3.10.8 Sending Task Attachments with Email Notifications

You can send task attachments with email notifications.

To send task attachments with email notifications:

1. In the **Notification** section, click the **Advanced** tab.
2. Select **Send task attachments with email notifications**.

27.3.10.9 Sending Email Notifications to Groups and Application Roles

You can send email notifications to groups and application roles to which tasks are assigned.

To send email notifications to groups and application roles:

1. In the **Notification** section, click the **Advanced** tab.
2. From the **Group notification configuration** list, select one of the following options.
 - **Send individual emails**

Each user in the group or application role receives an individual email notification. This is the default selection.

In addition, the **Use separate task forms based on locale** checkbox is automatically selected.

 - When selected, this sends individual emails with a separate task form based on the language locale.
 - When not selected, this sends individual emails and reuses (shares) the task form.
 - **Send one email containing all user addresses**

A shared notification email is generated once for a user locale in a group or application role, thereby saving time in notification email content generation. The email is sent to all users in the group or application role.

Notes:

- Since all (or a subset of) users receive the same email, the users in the group or application role are expected to have the same privilege. This ensures that the user does not see task details to which they are not entitled.
 - When sending one email to all users, the maximum number of characters allowed in the address field is 2000. If the limit is exceeded, email is sent to only those user addresses contained within the maximum limit.
-

27.3.10.10 Customizing Notification Headers

Custom notification headers are used to specify name and value pairs to identify key fields within the notification. These entries can be used by users to define delivery preferences for their notifications. For example:

You can set **Name** to **ApprovalType** and **value** to **Expense** or **Name** to **Priority** and **value** to **High**.

Users can then specify delivery preferences in Oracle BPM Worklist. These preferences can be based on the contents of the notification.

Note that the rule-based notification service is *only* used to identify the preferred notification channel to use. The address for the preferred channel is still obtained from the identity service.

To customize notification headers:

1. In the **Notification** section, click the **Advanced** tab.
2. Expand **Notification Header Attributes**.
3. Add name and pair value parameters by name or XPath expression.

For more information about preferences, see the following sections:

- [Section 31.2.8, "How to Send Inbound and Outbound Attachments"](#)
- [Section 31.2.14, "How to Create Custom Notification Headers"](#)
- [Part XI, "Using Oracle User Messaging Service"](#)

27.3.11 How to Specify Access Policies and Task Actions on Task Content

You can specify access rules on task content and actions to perform on that content.

27.3.11.1 Specifying Access Policies on Task Content

You can specify access rules that determine the parts of a task that participants can view and update. Access rules are enforced by the workflow service by applying rules on the task object during the retrieval and update of the task.

Note: Task content access rules and task actions access rules exist independently of one another.

27.3.11.1.1 Introduction to Access Rules Access rules are computed based on the following details:

- Any attribute configured with access rules declines any permissions for roles not configured against it. For example, assume you configure the payload to be read by assignees. This action enables *only* assignees and nobody else to have read permissions. No one, including assignees, has write permissions.
- Any attribute not configured with access rules has *all* permissions.
- If any payload message attribute is configured with access rules, any configurations for the payload itself are ignored due to potential conflicts. In this case, the returned map by the API does not contain any entry for the payload. Write permissions automatically provide read permissions.
- If only a subset of message attributes is configured with access rules, all message attributes not involved have all permissions.
- Only comments and attachments have add permissions.
- Write permissions on certain attributes are meaningless. For example, write permissions on history do not grant or decline any privileges on history.
- The following date attributes are configured as one in the Human Task Editor. The map returned by `TaskMetadataService.getVisibilityRules()` contains one key for each. Similarly, if the participant does not have read permissions on DATES, the task does not contain any of the following task attributes:
 - `START_DATE`
 - `END_DATE`
 - `ASSIGNED_DATE`
 - `SYSTEM_END_DATE`
 - `CREATED_DATE`
 - `EXPIRATION_DATE`
 - `ALL_UPDATED_DATE`
- The following assignee attributes are configured as one in the Human Task Editor. The map returned by `TaskMetadataService.getVisibilityRules()` contains one key for each of the following. Similarly, if the participant does not have read permissions on ASSIGNEES, the task does not contain any of the following task attributes:
 - `ASSIGNEES`
 - `ASSIGNEE_USERS`
 - `ASSIGNEE_GROUPS`
 - `ACQUIRED_BY`
- Mapped attributes do not have individual representation in the map returned by `TaskMetadataService.getVisibilityRules()`.
- All message attributes in the map returned by `TaskMetadataService.getVisibilityRules()` are prefixed by `ITaskMetadataService.TASK_VISIBILITY_ATTRIBUTE_PAYLOAD_MESSAGE_ATTR_PREFIX (PAYLOAD)`.

An application can also create pages to display or not display task attributes based on the access rules. This can be achieved by retrieving a participant's access rules by

calling the API on
`oracle.bpel.services.workflow.metadata.ITaskMetadataService`.
[Example 27–1](#) provides details.

Example 27–1 API Call

```
public Map<String, IPrivilege> getTaskVisibilityRules(IWorkflowContext context,
                                                    String taskId)
    throws TaskMetadataServiceException;
```

For more information about this method, see *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*.

27.3.11.1.2 Specifying User Privileges for Acting on Task Content You can specify the privileges that specific users (such as the task creator or owner) have for acting on specific task content (such as a payload).

To specify user privileges for acting on task content:

1. Click the **Access** tab.
2. Click the **Content** tab.
3. Select the task content for which to specify access privileges, as shown in [Figure 27–64](#).

Figure 27–64 Configure Task Content Access

Set access levels for each content item: ☒ Coarse grained ☐ Fine grained Reset

Content	Admin	Approvers	Assignees	Creator	Owner	Reviewers
Assignees	Read	Read	Read	Read	Read	Read
Attachments	Read	Read	Write	Write	Write	Write
Comments	Read	Read	Write	Write	Write	Write
Dates	Read	Read	Read	Read	Read	Read
Flexfields	Read	Read	Write	Write	Write	Read
History	Read	Read	Read	Read	Read	Read

Signature policy: None

Specify Restricted Assignment Configure Restricted Assignments...

4. Assign privileges (read, write, or no access) to users to act upon task content. Note that a user cannot be assigned a privilege above their highest level. For example, an **ADMIN** user cannot be assigned write access on the **PAYLOAD** task content. [Table 27–17](#) shows the maximum privilege each user has on task content.

Table 27–17 Highest Privilege Levels for Users of Task Content

Task Content	Individual with Read Access	Individual with Write Access
Assignees	Admin, Approvers, Assignees, Creator, Owner, Reviewers	--
Attachments	Admin, Approvers	Assignees, Creator, Owner, Reviewers

Table 27–17 (Cont.) Highest Privilege Levels for Users of Task Content

Task Content	Individual with Read Access	Individual with Write Access
Comments	Admin, Approvers	Assignees, Creator, Owner, Reviewers
Dates	Admin, Approvers, Assignees, Creator, Owner, Reviewers	--
Flexfields	Admin, Approvers, Reviewers	Assignees, Creator, Owner
History	Admin, Approvers, Assignees, Creator, Owner, Reviewers	--
Payload	Admin, Approvers, Reviewers	Assignees, Creator, Owner
Reviewers	Admin, Approvers, Assignees, Creator, Owner, Reviewers	--
Payload elements	Inherited from payload	Inherited from payload

For example, if you accept the default setting of **ASSIGNEES**, **CREATOR**, and **OWNER** with write access, **ADMIN**, **APPROVERS**, and **REVIEWERS** with read access, and **PUBLIC** with no access to the **PAYLOAD** task content, the dialog appears as shown in [Figure 27–64](#).

5. Select the method for displaying task content in this dialog. Note that choosing the currently unselected option causes all settings to reset to their default values.

- **Coarse grained** (default)

Displays the task content as a whole (for example, displays only one payload or reviewer).

- **Fine grained**

Displays the content as individual elements (for example, displays all payloads (such as **p1**, **p2**, and **p3**) and all reviewers assigned to this task (such as **jstein**, **wfaulk**, and **cdickens**).

Note: Access rules are always applied on top of what the system permits, depending on who is performing the action and the current state of the task.

27.3.11.1.3 Specifying Actions for Acting Upon Tasks You can specify the actions (either access or no access) that specific users (such as the task creator or owner) have for acting on the task content (such as a payload) that you specified in the Configure Task Content Access dialog.

To specify actions for acting upon tasks:

1. Click the **Access** tab.
2. Click the **Actions** tab.
3. Select the task action for which to specify users, as shown in [Figure 27–65](#).

Figure 27–65 Selection of Add Action Access Rule

Check action boxes to permit access: ☒ Coarse grained ☐ Fine grained [Reset](#)

Actions	Admin	Approvers	Assignees	Creator	Owner	Reviewers
APPROVE	☐	☐	☒	☐	☒	☐
REJECT	☐	☐	☒	☐	☒	☐
Acquire	☐	☐	☒	☐	☒	☐
Adhoc Route	☐	☐	☒	☐	☒	☐
Delegate	☐	☐	☒	☐	☐	☐
Delete	☒	☐	☒	☒	☒	☐

Signature policy: None

Specify Restricted Assignment [Configure Restricted Assignments...](#)

4. Select if participants can or cannot perform the selected actions.
5. Select the method for displaying task actions in this dialog. Note that choosing the currently unselected option causes all settings to reset to their default values.
 - **Coarse grained** (default)
Displays the task actions as a whole (for example, displays only one approval or rejection).
 - **Fine grained**
Displays the content actions as individual elements. (for example, displays all approvals or rejections).

27.3.12 How to Specify a Workflow Digital Signature Policy

Digital signatures provide a mechanism for the nonrepudiation of digitally-signed human tasks. This ability to mandate that a participant acting on a task signs the details and their action before the task is updated ensures that they cannot repudiate it later.

Note: If digital signatures are enabled for a task, actionable emails are not sent during runtime. This is the case even if actionable emails are enabled during design time.

To specify a workflow digital signature policy:

1. Click the **Access** tab.
2. From the **Signature Policy** list, select **Configure Policy**, as shown in [Figure 27–66](#).

Figure 27–66 Digital Signatures

Actions	ADMIN	APPROVERS	ASSIGNEES	CREATOR	OWNER	REVIEWERS
APPROVE	☐	☐	☒	☒	☒	☐
REJECT	☒	☒	☒	☒	☒	☒
ACQUIRE	☒	☐	☒	☐	☒	☐
ADHOC_R...	☐	☐	☒	☐	☒	☐
DELEGATE	☐	☐	☒	☐	☐	☐

3. Specify the signature policy for task participants to use:

- **No signature required**

Participants can send and act upon tasks without providing a signature. This is the default policy.

- **Password required**

Participants specify a signature before sending tasks to the next participant. Participants must reenter their password while acting on a task. The password is used to generate the digital signature. A digital signature authenticates the identity of the message sender or document signer. This ensures that the original content of the sent message is unchanged.

- **Digital certificate required**

Participants must possess a digital certificate for the nonrepudiation of digitally-signed human tasks. A digital certificate establishes the participant's credentials. It is issued by a certification authority (CA). It contains the following:

- Your name
- A serial number
- Expiration dates
- A copy of the certificate holder's public key (used for encrypting messages and digital signatures)
- Digital signature of the certificate-issuing authority so that message authenticity can be established

The CA names and CA CRL and URLs of the issuing authorities must be configured separately.

4. Click **OK**.

For more information, see [Section 31.1.10, "Evidence Store Service and Digital Signatures."](#)

27.3.12.1 Specifying a Certificate Authority

To use digital signatures, you must specify CAs you consider trustworthy in the System MBean Browser in Oracle Enterprise Manager Fusion Middleware Control. Only certificates issued from such CAs are considered valid by human workflow.

To specify a certificate authority:

1. From the **SOA Infrastructure** menu, select **Administration > System MBean Browser**.
2. Select **Application Defined MBeans > oracle.as.soainfra.config > Server: *server_name* > WorkflowConfig > human.workflow**.
3. Click the **Operations** tab.
4. Click **AddTrustedCA**.
5. In the **Value** fields for **CaName** and **CaURL**, specify appropriate values.
6. Click **Invoke**.
7. Click **Return**.

You must validate these values before using them.

27.3.13 How to Specify Restrictions on Task Assignments

You can restrict the users to which a task can be reassigned or routed by using a callback class.

The user community seeded in a typical LDAP directory can represent the whole company or division. However, it may be necessary at times to limit the potential list of users to associate with a task based on the scope or importance of the task or associated data. For example, in a large company with thousands of users, only a few people have the ability to approve and create purchase orders. Specifically for such tasks, the users that can be chosen for ad hoc routing and reassignment should not be the whole company. Instead, only a few users who are relevant or have the right privilege should be chosen. This can be achieved by the restricted assignment functionality. This is implemented as a callback class that can implement the logic to choose the right set of users dynamically based on the task object that is passed containing the instance data.

To specify restrictions on task assignments:

1. In the **Access** section, click **Configure Restricted Assignments**.
The Configure Restricted Assignment dialog appears.
2. Enter the class name. The class must implement the `oracle.bpel.services.workflow.task.IRestrictedAssignmentCallback` interface.
3. Click the **Add** icon to add name and value pairs for the property map passed to invoke the callback.
4. Click **OK**.

27.3.14 How to Specify Java or Business Event Callbacks

You can specify Java or business event callbacks.

27.3.14.1 Specifying Callback Classes on Task Status

You can register callbacks for the workflow service to call when a particular stage is reached during the lifecycle of a task. Two types of callbacks are supported:

- **Java callbacks:** The callback class must implement the interface `oracle.bpel.services.workflow.task.IRoutingSlipCallback`. Make the callback class available in the class path of the server.
- **Business event callbacks:** You can have business events raised when the state of a human task changes. You do not need to develop and register a Java class. The caller implements the callback using an Oracle Mediator service component to subscribe to the applicable business event to be informed of the current state of an approval transaction.

To specify callback classes on task status:

1. Click the **Events** tab.

The following callbacks are available for selection:

- **OnAssigned**
Select if the callback class must be called on any assignment change, including standard routing, reassignment, delegation, escalation, and so on. If a callback is required when a task has an outcome update (that is, one of the approvers in a chain approves or rejects the task), this option must be selected.
 - **OnUpdated**
Select if the callback class must be called on any update (including payload, comments, attachments, priority, and so on).
 - **OnCompleted**
Select if the callback class must finally be called when the task is completed and control is about to be passed to the initiator (such as the BPEL process initiating the task).
 - **OnStageCompleted**
Select if the callback class must be called to enable business event callbacks in a human workflow task. When the event is raised, it contains the name of the completed stage, the outcome for the completed stage, and a snapshot of the task when the callback is invoked.
 - **OnSubtaskUpdated**
Select if the callback class must be called on any update (including payload, comments, attachments, priority, and so on) on a subtask (one of the tasks in a parallel-and-parallel scenario).
2. See the following section based on the type of callback to perform.
 - [Section 27.3.14.1.1, "Specifying Java Callbacks"](#)
 - [Section 27.3.14.1.2, "Specifying Business Event Callbacks"](#)

27.3.14.1.1 Specifying Java Callbacks

To specify Java callbacks:

1. In the **State** column of the **Events** section, select a task state.
2. In the **Java Class** column, click the empty field to enter a value. This value is the complete class name of the Java class that implements

`oracle.bpel.services.workflow.task.IRoutingSlipCallback`.
 Figure 27–67 provides details.

Figure 27–67 Callback Details Dialog with Java Selected

State	Java Class	Trigger Workflow Event
OnAssigned		☒
OnUpdated		☐
OnCompleted		☐
OnStageCompleted		☐
OnSubtaskUpdated		☐

☐ Allow task and routing customization in BPEL callbacks

☐ Disable BPEL callbacks

3. Click OK.

27.3.14.1.2 Specifying Business Event Callbacks

To specify business event callbacks:

1. In the **State** column of the **Events** section, select a task state.
2. Leave the **Java Class** field empty.
3. Select the **Trigger Workflow Event** checkbox. This action disables the **Java Class** column, as shown in Figure 27–68. Each callback, such as **OnAssigned**, corresponds to a business event point. When a business event is fired, the event details contain the task object and a set of properties that are populated based on the context of the event being fired.

Figure 27–68 Callback Details Dialog with Business Events Selected

State	Java Class	Trigger Workflow Event
OnAssigned		☒
OnUpdated		☐
OnCompleted		☐
OnStageCompleted		☐
OnSubtaskUpdated		☐

☐ Allow task and routing customization in BPEL callbacks

☐ Disable BPEL callbacks

A preseeded, static event definition language (EDL) file (`JDev_Home\jdeveloper\integration\seed\soa\shared\workflow\HumanTas`

`kEvent.edl`) provides the list of available business events to which to subscribe. These business events correspond to the callbacks you select in the Callback Details dialog. You must now create an Oracle Mediator service component in which you reference the EDL file and subscribe to the appropriate business event.

Note: A file-based MDS connection is required so that the EDL file can be located. The location for the file-based MDS is `JDev_Home\jdeveloper\integration\seed`.

4. Create an Oracle Mediator service component in the same or a different SOA composite application that can subscribe to the event.
5. In the **Template** list during Oracle Mediator creation, select **Subscribe to Events**.
6. Click the **Add** icon to subscribe to a new event.
7. To the right of the **Event Definition** field, click the **Browse** icon to select the EDL file.

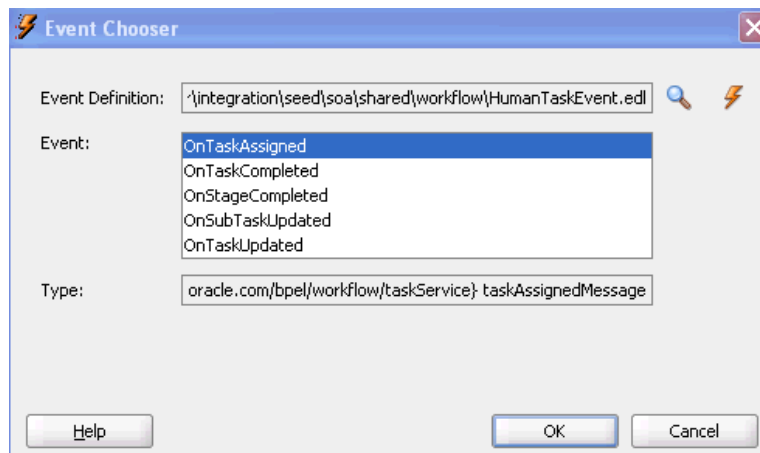
The SOA Resource Browser dialog appears.

8. Select the previously created file-based MDS connection.
9. From the list at the top, select **Resource Palette**.
10. Select **SOA > Shared > Workflow > HumanTaskEvent.edl**.
11. Click **OK**.

The Event Chooser is now populated with EDL file business events available for selection.

12. In the **Event** field, select the event to which to subscribe. [Figure 27–69](#) provides details.

Figure 27–69 Event Callbacks



You can have multiple human tasks available for subscribing to the event. For example, assume you performed the following:

- Configured a human task named TaskA to subscribe to the event (for example, **OnAssigned**)
- Configured a human task named TaskB to subscribe to the same event

To distinguish between events for TaskA and TaskB and ensure that an event is processed only by the intended Oracle Mediator, you can add a static routing filter:

```
xpath20:compare (med:getComponentName () , 'TaskA')
```

This only invokes this routing when the sending component is TaskA.

13. If the EDL file was *not* selected from the file-based MDS connection, accept to import the dependent XSD files when prompted, and click **OK**. If the EDL file was selected from the file-based MDS connection, you are not prompted.

The Oracle Mediator service component is now populated with the business event to which to subscribe. You can also subscribe to other business events defined in the same EDL file now or at a later time.

See the following documentation for additional details about business events and callbacks:

- [Chapter 38, "Using Business Events and the Event Delivery Network"](#) for specific details about business events
- Sample workflow-116-WorkflowEventCallback, which is available from the Oracle Technology Network:

<https://soasamples.samplecode.oracle.com>

27.3.15 How to Specify Task and Routing Customizations 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 can use the **Allow task and routing customization in BPEL callbacks** option.

Make sure to manually refresh the BPEL diagram for this callback setting.

To specify task and routing customizations in BPEL callbacks:

1. In the **Events** section, select the **Allow task and routing customization in BPEL callbacks** checkbox.
2. Return to Oracle BPEL Designer.
3. Open the task activity dialog.
4. Click **OK**.

This creates the while, pick, and `onMessage` branch of a pick activity for BPEL callback customizations inside the task scope activity.

For more information about specifying task and routing customizations, see [Section 27.4.5.1, "Invoking BPEL Callbacks."](#)

27.3.16 Disabling BPEL Callbacks

A user talk activity (in Oracle BPEL Designer) has an invoke activity followed by a receive or pick activity. Deselecting the **Disable BPEL callbacks** checkbox enables you to invoke the task service without waiting for a reply.

To disable BPEL callbacks:

1. In the **Events** section, deselect the **Disable BPEL callbacks** checkbox.

2. Click OK.

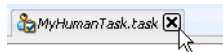
27.3.17 How to Exit the Human Task Editor and Save Your Changes

You can save your human task changes at any time. The task can be re-edited at a later time by double-clicking the metadata task configuration `.task` file in the Application Navigator.

To exit the Human Task Editor and save your changes:

1. From the **File** main menu, select **Save** or click the X sign shown in [Figure 27-70](#) to close the `.task` metadata task configuration file.

Figure 27-70 File Closure



2. If you click the X sign, select **Yes** when prompted to save your changes.

27.4 Associating the Human Task Service Component with a BPEL Process

To associate the human task service component created in the SOA Composite Editor with a BPEL process, follow these instructions. When association is complete, a task service partner link is created in Oracle BPEL Designer. The task service exposes the operations required to act on a task.

For more information about creating a human task, see [Section 27.3, "Creating the Human Task Definition with the Human Task Editor."](#)

27.4.1 How to Associate a Human Task with a BPEL Process

There are two ways to associate a human task service component with a BPEL process:

- If you have created a human task service component in the SOA composite application, drag a human task activity into the BPEL process in Oracle BPEL Designer. Then, select the existing human task service component from the **Task Definition** list of the Create Human Task dialog. You can then specify the task title, initiator, parameter values, and other values.
- If you have not created a human task service component, drag the human task activity into the BPEL process in Oracle BPEL Designer. Then, click the **Add** icon to the right of the **Task Definition** list in the Create Human Task dialog. This action enables you to specify the name of the new human task service component, the parameters, and the outcomes. The Human Task Editor then opens for you to design the remaining task metadata. After design completion, close the Human Task Editor.

To associate a human task with a BPEL process:

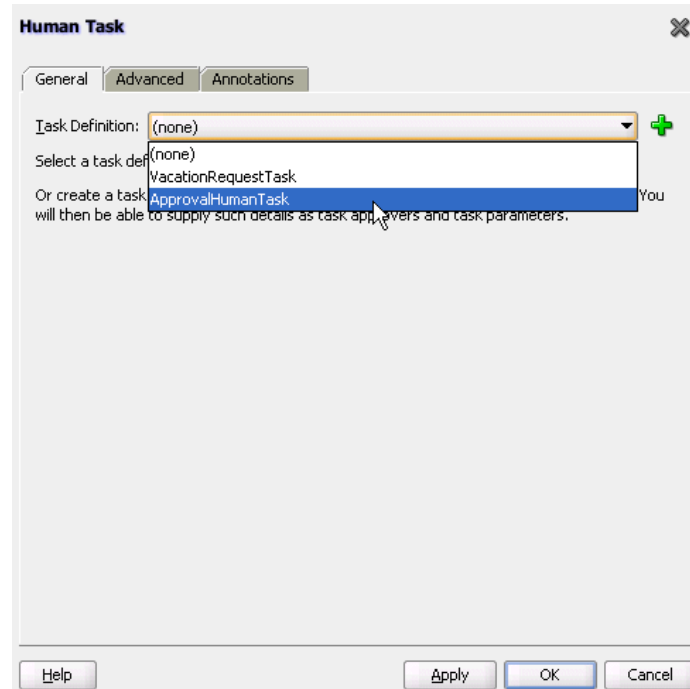
1. Go to the SOA Composite Editor.
2. Double-click the BPEL process service component with which to associate the `.task` file of the human task service component.
3. In the Component Palette, expand **SOA Components**.
4. Drag a new **Human Task** activity into the BPEL process.

5. Double-click the **Human Task** activity.

The Human Task dialog appears.

6. From the **Task Definition** list of the **General** tab, select the human task, as shown in [Figure 27-71](#).

Figure 27-71 Task Definition List Selection



The .task file of the human task service component is associated with the BPEL process.

Note: After you complete association of your human task activity with a BPEL process and close the Create Human Task dialog, you can always re-access this dialog by double-clicking the human task activity in Oracle BPEL Designer.

27.4.2 What You May Need to Know About Deleting a Wire Between a Human Task Service Component and a BPEL Process

If you delete the wire between a BPEL process and the human task service component that it invokes, the invoke activity of the human workflow is deleted from the BPEL process. However, the **taskSwitch** switch activity for taking action (contains the approve, reject, and otherwise task outcomes) is still there. This is by design for the following reasons:

- The switch activity contains user-entered BPEL code.
- The switch can be reused if the intention for deleting the wire is only to point to another human task.
- Deleting the switch is a single-step action.

If you then drag and drop a human task service component into the BPEL process to use the same **taskSwitch** switch activity, a new **taskSwitch** switch activity is created.

You then have two switch activities in the BPEL process with the same name. To determine which one to delete, you must go into the approve, reject, and otherwise task outcomes of the **taskSwitch** switch activities to determine which is the older, modified switch and which is the newer switch.

27.4.3 How to Define the Human Task Activity Title, Initiator, Priority, and Parameter Variables

Figure 27-72 shows the **General** tab that displays after you select the human task.

Figure 27-72 Human Task — General Tab (After Selection)

The image shows a software dialog box titled "Human Task" with a close button (X) in the top right corner. It has three tabs: "General", "Advanced", and "Annotations", with "General" being the active tab. In the "General" tab, there is a "Task Definition" dropdown menu showing "ApprovalHumanTask" with a green plus icon to its right. Below this is a "Task Title" text field with a placeholder text "Enter text to override default value from .task" and a small icon to its right. Underneath the "Task Title" field is a sample text: "E.g., Vacation Request for <%bpws:getVariableData(...)%>". Below the "Task Title" field is an "Initiator" text field with a small icon to its right. Below the "Initiator" field is a "Priority" dropdown menu showing the value "3". At the bottom of the dialog is a large empty rectangular area labeled "Task Parameters" and "BPEL Variable". At the very bottom of the dialog are four buttons: "Help", "Apply", "OK", and "Cancel".

The **General** tab of the Human Task activity enables you to perform the tasks shown in Table 27-18:

Table 27-18 Human Task - General Tab

For this Field...	See...
Task Title	Section 27.4.3.1, "Specifying the Task Title"
Initiator	Section 27.4.3.2, "Specifying the Task Initiator and Task Priority"
Priority	
Task Parameters	Section 27.4.3.3, "Specifying Task Parameters"

27.4.3.1 Specifying the Task Title

The title displays the task in Oracle BPM Worklist during runtime. This is a mandatory field. Your entry in this field overrides the task title you entered in the **Task Title** field of the **General** section of the Human Task Editor described in [Section 27.3.4.1, "Specifying a Task Title."](#)

To specify the task title:

1. In the **Task Title** field of the **General** tab, enter the task title through one of the following methods:
 - Enter the title manually.
 - Click the icon to the right of the field to display the Expression Builder dialog to dynamically create the title.

You can also combine static text and dynamic expressions in the same title. To include dynamic text, place your cursor at the appropriate point in the text and click the icon on the right to invoke the Expression Builder dialog.

27.4.3.2 Specifying the Task Initiator and Task Priority

You can specify a task initiator. The initiator is the user who initiates a task. The initiator can view their created tasks from Oracle BPM Worklist and perform specific tasks, such as withdrawing or suspending a task.

To specify the task initiator and task priority:

1. To the right of the **Initiator** field of the **General** tab, enter the initiator (for example, jcooper) or click the icon to display the Expression Builder dialog for dynamically specifying an initiator. This field is optional. If not specified, the initiator defaults to the task owner specified on the **Advanced** tab of the Human Task dialog. The initiator defaults to bpeadmin if a task owner is also not specified.
2. From the **Priority** list, select a priority value between 1 (the highest) and 5. This field is provided for user reference and does not make this task a higher priority during runtime. Use the priority to sort tasks in Oracle BPM Worklist. This priority value overrides the priority value you select in the **Priority** list of the **General** section of the Human Task Editor.

For more information about specifying the priority in the Human Task Editor, see [Section 27.3.4.1, "Specifying a Task Title."](#)

27.4.3.3 Specifying Task Parameters

The task parameter table shown in [Figure 27-73](#) displays a list of task parameters after you complete the **Task Title** and **Initiator** fields.

Figure 27–73 Task Parameter Table

Human Task

General Advanced Annotations

Task Definition: ApprovalHumanTask

Task Title: `getVariableData('inputVariable','payload','/client:process/client:orderId')%>`
 E.g., Vacation Request for <%bpws:getVariableData(...)%>

Initiator: `nerInfoVO1CustomerInfoVOCriteriaResponse/ns6:result/ns4:LastName'%>`

Priority: 3

Task Parameters	BPEL Variable
orderId	

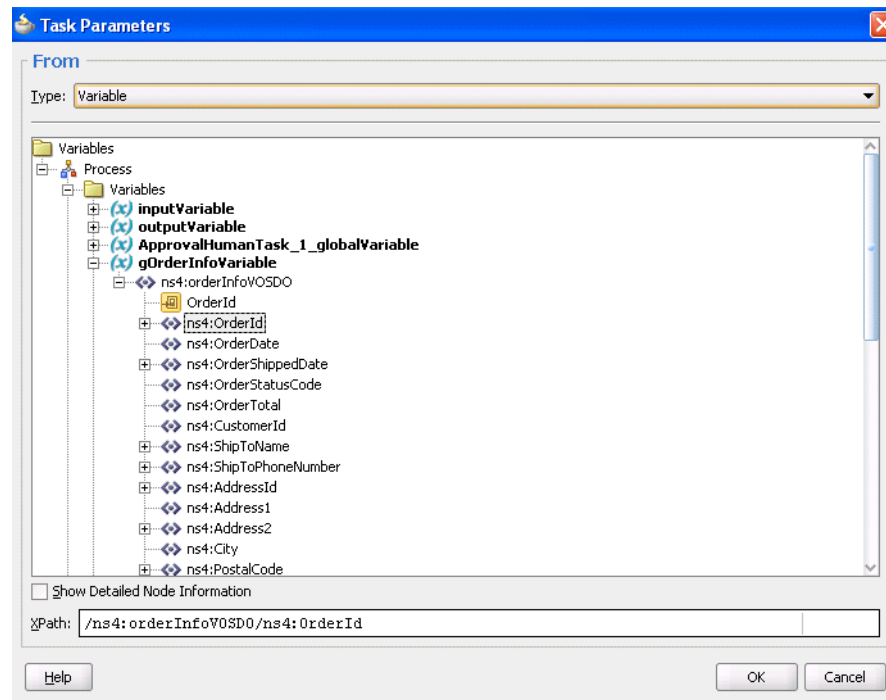
Help Apply OK Cancel

To specify task parameters:

1. In the **BPEL Variable** column, double-click the **dots** to map the task parameter to the BPEL variable. You must map only the task parameters that carry input data. For output data that is filled in from Oracle BPM Worklist, you do not need to map the corresponding variables.

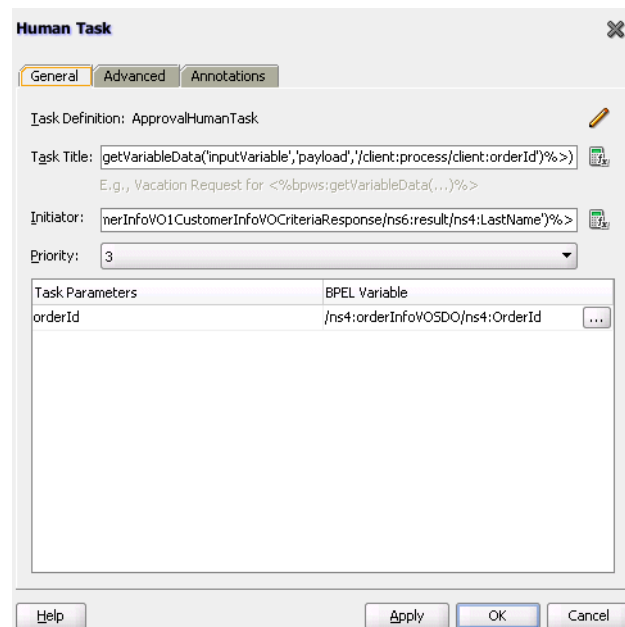
The Task Parameters dialog appears.

2. Expand the **Variables** tree shown in [Figure 27–74](#) and select the appropriate task variable.

Figure 27–74 Variables Tree

3. Click **OK**.

The Human Task dialog shown in [Figure 27–75](#) appears as follows.

Figure 27–75 Human Task Dialog

4. To define advanced features for the human task activity, click the **Advanced** tab and go to [Section 27.4.4, "How to Define the Human Task Activity Advanced Features."](#) Otherwise, click **OK** to close the Human Task dialog.

27.4.4 How to Define the Human Task Activity Advanced Features

Figure 27–76 shows the **Advanced** tab.

Figure 27–76 Create Human Task — Advanced Tab

The screenshot shows the 'Human Task' dialog box with the 'Advanced' tab selected. The 'Scope Name' field contains 'ApprovalHumanTask_1' and the 'Global Task Variable Name' field contains 'ApprovalHumanTask_1_globalVariable'. The 'Owner' field contains 'jcooper'. The 'Identification Key', 'Identity Context', and 'Application Context' fields are empty. The 'Include task history from' checkbox is unchecked. The 'Clear old payload and recreate' radio button is selected. The 'OK' button is highlighted.

The **Advanced** tab of the Human Task activity enables you to perform the tasks shown in Table 27–19:

Table 27–19 Human Task - Advanced Tab

For this Field...	See...
Scope Name	Section 27.4.4.1, "Specifying a Scope Name and a Global Task Variable Name"
Global Task Variable Name	
Owner	Section 27.4.4.2, "Specifying a Task Owner"
Identification Key	Section 27.4.4.3, "Specifying an Identification Key"
Identity Context	Section 27.4.4.4, "Specifying an Identity Context"
Application Context	Section 27.4.4.5, "Specifying an Application Context"
Include task history from	Section 27.4.4.6, "Including the Task History of Other Human Tasks"

27.4.4.1 Specifying a Scope Name and a Global Task Variable Name

You are automatically provided with default scope and global task variable names during human task activity creation. However, you can specify custom names that are used to name the scope and global variable during human task activity creation.

To specify a scope name and a global task variable name:

1. In the **Scope Name** field of the **Advanced** tab, enter the name for the BPEL scope to be generated.

This BPEL scope encapsulates the entire interaction with the workflow service and BPEL variable manipulation.

2. In the **Global Task Variable Name** field of the **Advanced** tab, enter the global task variable name.

This is the name of the BPEL task variable used for the workflow interaction.

27.4.4.2 Specifying a Task Owner

The task owner can view tasks belonging to business processes they own and perform operations on behalf of any of the task assignees. Additionally, the owner can also reassign, withdraw, or escalate tasks.

If you do not specify a task initiator on the **General** tab of the Human Task dialog, it defaults to the owner specified here.

To specify a task owner:

1. In the **Owner** field of the **Advanced** tab, enter the task owner name or click the icon to the right to use the Expression Builder to dynamically specify the owner of this task.

27.4.4.3 Specifying an Identification Key

The identification key can be used as a user-defined ID for the task. For example, if the task is meant for approving a purchase order, the purchase order ID can be set as the identification key of the task. Tasks can be searched from Oracle BPM Worklist using the identification key. This attribute has no default value.

To specify an identification key:

1. In the **Identification Key** field of the **Advanced** tab, enter an optional identification key value.

27.4.4.4 Specifying an Identity Context

The identity realm name is used for the task when multiple realms are configured. You cannot have assignees from multiple realms working on the same task. This field is required if you are using multiple realms.

To specify an identity context

1. In the **Identity Context** field of the **Advanced** tab, enter a value.

27.4.4.5 Specifying an Application Context

The stripe name of the application contains the application roles used in the task.

To specify an application context

1. In the **Application Context** field of the **Advanced** tab, enter a value.

27.4.4.6 Including the Task History of Other Human Tasks

This feature enables one human task to be continued with another human task. There are many scenarios in which you have related tasks in a single BPEL process. For example, assume you have the following:

- A procurement process to obtain a manager's approval for a computer
- Several BPEL activities in between
- Another task for the IT department to buy the computer

The participant of the second task may want to see the approval history, comments, and attachments created when the manager approved the purchase. You can link these different tasks in the BPEL process by chaining the second task to the first task with this option.

For chained tasks, the title of the new task cannot be set from the task metadata (.task file). For example, assume existing Task A is chained with new task Task B, and Task B has a new title set in the Human Task Editor; this title is *not* recognized. Therefore, if the chained task requires a different title, it must be set in the task instance before calling the task service `reinitiate` operation. If a BPEL process is initiating the tasks, set the task title before the workflow service APIs are called. If a Java program is calling the workflow APIs programatically, then it must set the title.

To include the task history of other tasks:

1. Select the **Include task history from** checkbox of the **Advanced** tab to extend a previous workflow task in the BPEL process. Selecting this checkbox includes the task history, comments, and attachments from the previous task. This provides you with a complete end-to-end audit trail.

When a human task is continued with another human task, the following information is carried over to the new workflow:

- Task payload and the changes made to the payload in the previous workflow
- Task history
- Comments added to the task in the previous workflow
- Attachments added to the task in the previous workflow
- Due date

In the **Include task history from** list, all existing workflows are listed.

2. Select a particular human task to extend (continue) the selected human task.

For example, a hiring process is used to hire new employees. Each interviewer votes to hire or not hire a candidate. If 75% of the votes are to hire, then the candidate is hired; otherwise, the candidate is rejected. If the candidate is to be hired, an entry in the HR database is created and the human resources contact completes the hiring process. The HR contact also must see the interviewers and the comments they made about the candidate. This process can be modeled using a parallel participant type for the hiring. If the candidate is hired, a database adapter is used to create the entry in the HR database. After this action, a simple workflow can include the task history from the parallel participant type so that the hiring request, history, and interviewer comments are carried over. This simple workflow is assigned to the HR contact.

3. Select a payload to use:

- **Clear old payload and recreate**

This option is applicable when the payload attributes in the XML files of the human tasks involved in this extended workflow are different. For example, the payload attribute for the human task whose history you are including has three extra attributes than the payload of the other human task.

- **Use existing payload**

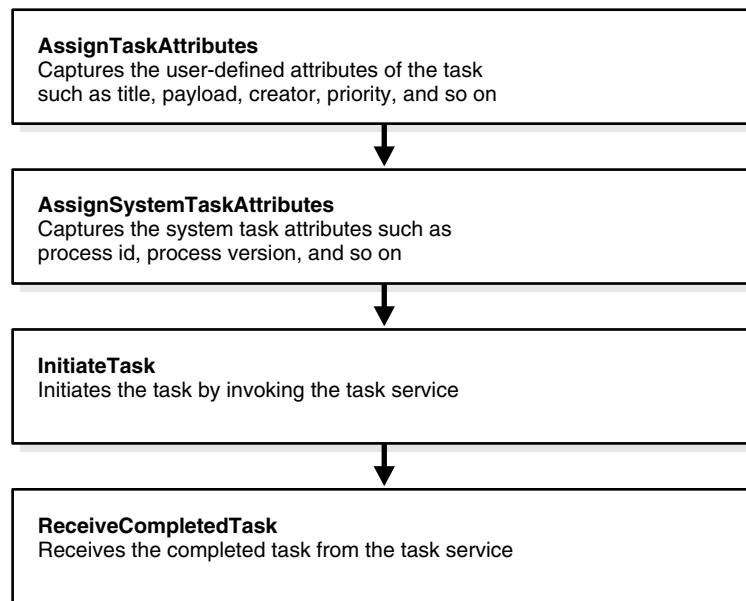
This option is applicable when the payload attributes in the XML files of the human tasks involved in this extended workflow are the same.

27.4.5 How to View the Generated Human Task Activity

When you have completed modeling the human task activity, the human task is generated in the designer.

Figure 27–77 shows how a workflow interaction is modeled. Figure 27–77 also illustrates the interaction when no BPEL callbacks are modeled. In this case, after a task is complete, the BPEL process is called back with the completed task. No intermediary events are propagated to the BPEL process instance. It is recommended that any user customizations be done in the first assign, AssignTaskAttributes, and that AssignSystemTaskAttributes not be changed.

Figure 27–77 Workflow Interaction Modeling



Click the **Expand** icon next to the human task activity in Oracle BPEL Designer to display its contents, as shown in Figure 27–78.

Figure 27–78 Expanding the Human Task Activity



27.4.5.1 Invoking BPEL Callbacks

If intermediary events must be propagated to the BPEL process instance, select the **Allow task and routing customization in BPEL callbacks** checkbox in the **Events** section of the Human Task Editor. When this option is selected, the workflow service invokes callbacks in the BPEL instance during each update of the task. The callbacks are listed in the `TaskService.wsdl` file and described as follows:

- `onTaskCompleted`
This callback is invoked when the task is completed, expired, withdrawn, or errored.
- `onTaskAssigned`

This callback is invoked when the task is assigned to a new set of assignees due to the following actions:

- Outcome update
- Skip current assignment
- Override routing slip

■ `onTaskUpdated`

This callback is invoked for any other update to the task that does not fall in the `onTaskComplete` or `onTaskAssigned` callback. This includes updates on tasks due to a request for information, a submittal of information, an escalation, a reassign, and so on.

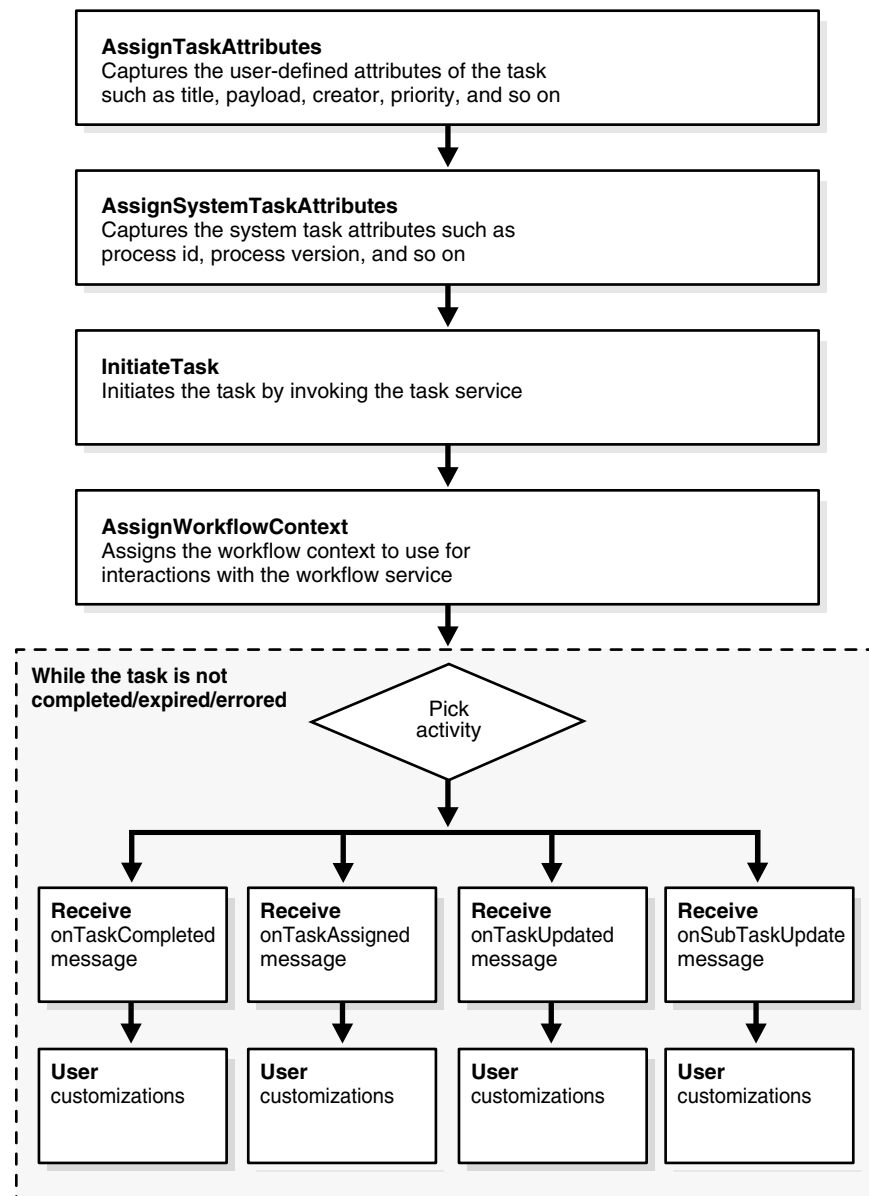
■ `onSubTaskUpdated`

This callback is invoked for any update to a subtask.

[Figure 27–79](#) shows how a workflow interaction with callbacks is modeled. After this task is initiated, a while loop is used to receive messages until the task is complete. The while loop contains a pick with four `onMessage` branches — one for each of the above-mentioned callback operations. The workflow interaction works fine even if nothing is changed in the `onMessage` branches, meaning that customizations in the `onMessage` branches are not required.

In this scenario, a workflow context is captured in the BPEL instance. This context can be used for all interaction with the workflow services. For example, 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 performed in the first assign, `AssignTaskAttributes`, and that `AssignSystemTaskAttributes` not be changed.

Figure 27–79 Workflow Interaction Modeling (with Callbacks)

27.4.6 What You May Need to Know About Changing the Generated Human Task Activity

If you must change a generated human task activity, note the following details:

- Do *not* modify the assign tasks that are automatically created in a switch activity when you add a human task to a BPEL process flow. Instead, add a new assign activity outside the switch activity.
- If the parameter passed into a human task is modified (for example, you change the parameter type in the Edit Task Parameter dialog), you must open the human task activity in the BPEL process flow and click OK to correct the references to the payload variable. Not doing so causes the parameter name to change and become uneditable.

If the task outcomes in the Human Task Editor are modified, you must edit the human task activity and click **OK**. The switch case is then updated based on the changes to the outcomes.

- If you make any changes to the translatable strings of the title or category of a task in the resource bundle, those changes do not appear in any instances of that task that are already initiated. However, they do appear in instances of that task that are initiated after you make the changes.

27.4.7 What You May Need to Know About Deleting a Partner Link Generated by a Human Task

Deleting a partner link that was generated by a human task (for example, *human_task_name.TaskService* in the **Partner Links** swimlane) causes the human task to become unusable. If you delete the partner link, you must delete the human task activity in Oracle BPEL Designer and start over again.

27.4.8 How to Define Outcome-Based Modeling

In many cases, the outcome of a task determines the flow of the business process. To facilitate modeling of the business logic, when a user task is generated, a BPEL switch activity is also generated with prebuilt BPEL case activities. By default, one case branch is created for each outcome selected during creation of the task. An otherwise branch is also generated in the switch to represent cases in which the task is withdrawn, expired, or in error.

27.4.8.1 Specifying Payload Updates

The task carries a payload in it. If the payload is set from a business process variable, then an assign activity with the name `copyPayloadFromTask` is created in each of the case and otherwise branches to copy the payload from the task back to its source. If the payload is expressed as other XPath expressions (such as `ora:getNodes(...)`), then this assign is not created because of the lack of a process variable to copy the payload back. If the payload does not require modification, then this assign can be removed.

27.4.8.2 Using Case Statements for Other Task Conclusions

By default, the switch activity contains case statements for the outcomes only. The other task conclusions are captured in the otherwise branch. These conclusions are as follows:

- The task is withdrawn.
- The task is in error.
- The task is expired.

If business logic must be added for each of these other conclusions, then case statements can be added for each of the preceding conditions. The case statements can be created as shown in the following BPEL segment. The XPath conditions for the other conclusions in the case activities for each of the preceding cases are shown in bold in [Example 27–2](#).

Example 27–2 XPath Conditions for Other Conclusions in the Case Activities

```
<switch name="taskSwitch">
  <case condition="bpws:getVariableData('SequentialWorkflowVar1',
'/task:task/task:state') = 'COMPLETED' and
```

```

bpws:getVariableData('SequentialWorkflowVar1', '/task:task/task:conclusion') =
'ACCEPT'">
    <bpelx:annotation>
        <bpelx:pattern>Task outcome is ACCEPT
        </bpelx:pattern>
    </bpelx:annotation>
    ...
</case>
<case condition=
"bpws:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') =
'WITHDRAWN'">
    <bpelx:annotation>
        <bpelx:pattern>Task is withdrawn
        </bpelx:pattern>
    </bpelx:annotation>
    ...
</case>
<case condition=
"bpws:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') =
'EXPIRED'">
    <bpelx:annotation>
        <bpelx:pattern>Task is expired
        </bpelx:pattern>
    </bpelx:annotation>
    ...
</case>
<case condition=
"bpws:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') =
'ERRORED'">
    <bpelx:annotation>
        <bpelx:pattern>Task is errored
        </bpelx:pattern>
    </bpelx:annotation>
    ...
</case>
<otherwise>
    <bpelx:annotation>
        <bpelx:pattern>Task is EXPIRED, WITHDRAWN or ERRORED
        </bpelx:pattern>
    </bpelx:annotation>
    ...
</otherwise>
</switch>

```

Designing Task Forms for Human Tasks

The human workflow service creates tasks for users to interact with the business process. Each task has two parts—the task metadata and the task form. The task form is used to display the contents of the task to the user's worklist.

Oracle BPM Worklist displays all worklist tasks that are assigned to a user or a group. When a worklist user drills down into a specific task, the task form renders the details of that task. For example, the task form for the Fusion Order Demo ApprovalHumanTask shows order information such as the order total and ship-to information.

This chapter describes how to design and customize task forms using ADF task flows in Oracle JDeveloper.

This chapter includes the following sections:

- [Section 28.1, "Introduction to the Task Form"](#)
- [Section 28.2, "Associating the Task Flow with the Task Service"](#)
- [Section 28.3, "Creating an ADF Task Flow Based on a Human Task"](#)
- [Section 28.4, "Creating a Task Form"](#)
- [Section 28.5, "Refreshing Data Controls When the Task XSD Changes"](#)
- [Section 28.6, "Securing the Task Flow Application"](#)
- [Section 28.7, "Creating an Email Notification"](#)
- [Section 28.8, "Deploying a Composite Application with a Task Flow"](#)
- [Section 28.9, "Displaying a Task Form in the Worklist"](#)
- [Section 28.10, "Displaying a Task in an Email Notification"](#)
- [Section 28.11, "Reusing the Task Flow Application with Multiple Human Tasks"](#)

For information about troubleshooting human workflow issues, see section "Human Workflow Troubleshooting" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

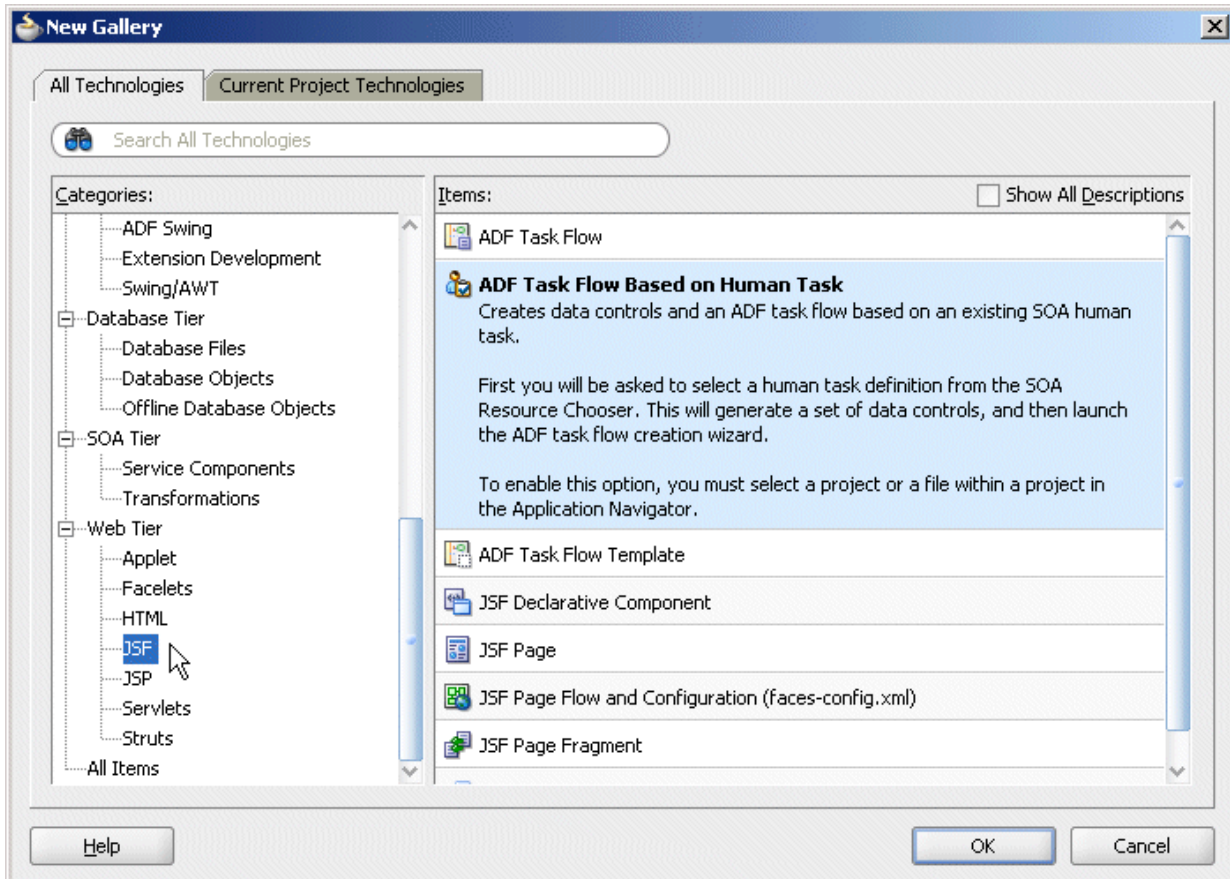
28.1 Introduction to the Task Form

If your SOA composite includes a human task, then you need a way for users to interact with the task. The integrated development environment of Oracle SOA Suite includes Oracle Application Development Framework (Oracle ADF) for this purpose. With Oracle ADF, you can design a task form that depicts the human task in the SOA composite.

The task form is a Java Server Page XML (.jspx) file that you create in the Oracle JDeveloper designer where you created the SOA composite containing the human task. Note that you must set the page encoding to UTF-8 in Oracle JDeveloper before creating the Java Server Page XML file. You can do this in Oracle JDeveloper by choosing **Tools > Preferences > Environment**, and selecting UTF-8 using the **Encoding** dropdown list.

Figure 28–1 shows the Oracle JDeveloper **ADF Task Flow Based on Human Task** option where you start creating a task form.

Figure 28–1 ADF Task Flow Based on a Human Task, in Oracle JDeveloper



This section contains these topics:

- [Section 28.1.1, "What You May Need to Know About Task Forms: Time Zone Conversion."](#)
-

28.1.1 What You May Need to Know About Task Forms: Time Zone Conversion

Time zone conversion is not automatic for datetime elements in the task payload when a task form is created. You must add the `<af:convertDateTime>` tag to enable time zone conversion on a datetime element. See any standard task header time label for an example. [Example 28–1](#) shows a sample header.

Example 28–1 Time Zone Conversion

```
<af:outputText value="#{bindings.createdDate.inputValue}"
```

```

        id="ot15">
<f:convertDateTime type="{pageFlowScope.dt}"
                  timeZone="{pageFlowScope.tz}"
                  dateStyle="{pageFlowScope.df}"
                  timeStyle="{pageFlowScope.tf}"/>
</af:outputText>

```

28.1.2 What You May Need to Know About Task Forms: Setting Formats for Date and Time

To specify formats for the date and time fields, add controlHints to the required fields. In the example below, I have used vacation request sample to add 3 additional date fields.

The following example uses a sample vacation request to add three additional date fields.

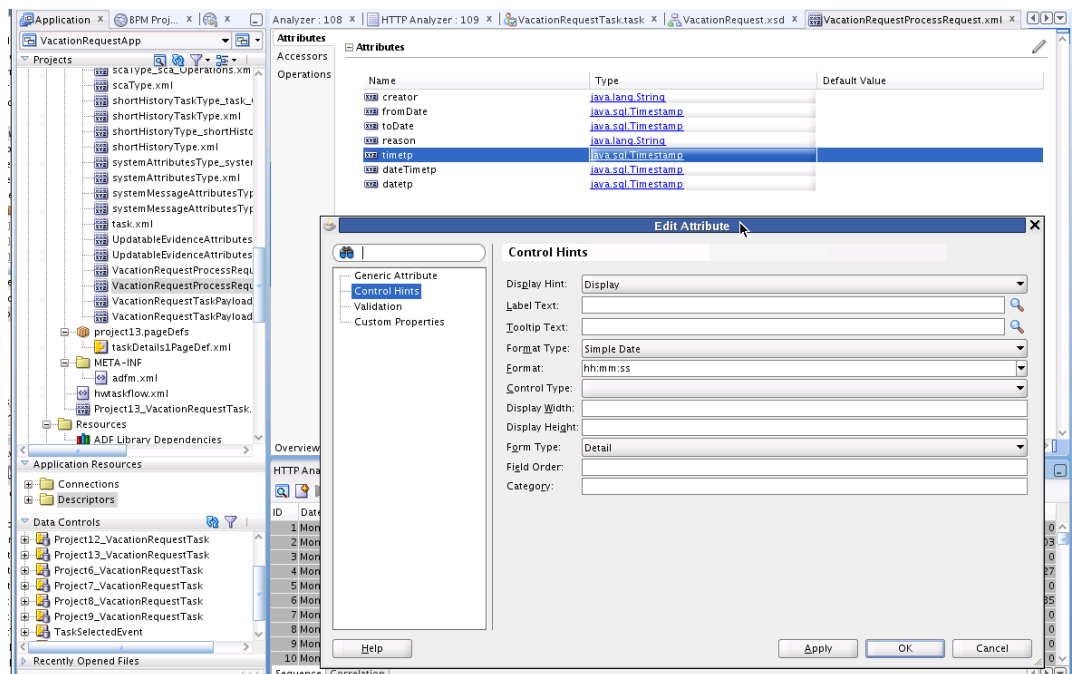
```

<element name="VacationRequestProcessRequest">
  <complexType>
    <sequence>
      <element name="creator" type="string"/>
      <element name="fromDate" type="date"/>
      <element name="toDate" type="date"/>
      <element name="reason" type="string"/>
      <element name="timeotp" type="time"/>
      <element name="dateTimetp" type="dateTime"/>
      <element name="datetp" type="date"/>
    </sequence>
  </complexType>
</element>

```

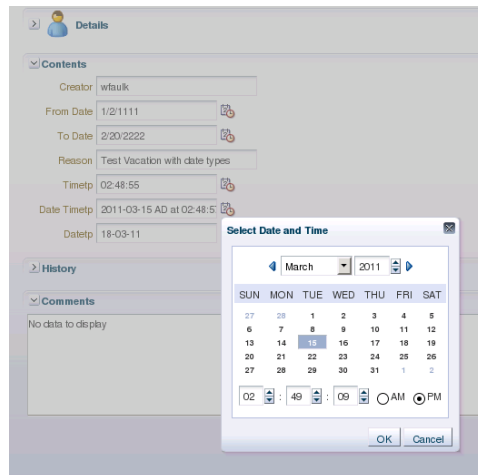
Within JDeveloper, you can choose the data control .xml file and add control hints. The example in [Figure 28-2](#) sets the time format for the field timeotp.

Figure 28-2 Setting the Time Format in JDeveloper



You can set the the format for the `dateTtp` and `dateTimetp` fields in the same way. When the application is deployed, the validation is based on the specified format. In Worklist Application, the Select Date and Time dialog box reflects the specified format as shown in [Figure 28–3](#).

Figure 28–3 Selecting Date and Time in Worklist Application



28.2 Associating the Task Flow with the Task Service

When you create an ADF task flow based on a human task, you must select a task metadata file to generate the data control. This data control is used to lay out the content on the page and connect to the workflow service engine at execution time to retrieve task content and act on tasks. See "Getting Started with ADF Task Flows" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* for more information.

The `hwtaskflow.xml` file is used to capture the details on connecting with the service engine. By default, it uses remote EJBs to connect to the workflow server. The SOA server URL and port are automatically determined by using WebLogic Server runtime MBeans. However, you can override these by explicitly specifying the URL and port information here.

Seed a user that has ORMI privileges so that the task details application can connect to the workflow service. You can seed this user by using Oracle Enterprise Manager Fusion Middleware Control.

28.3 Creating an ADF Task Flow Based on a Human Task

ADF task flows are used to model the user interface for the task details page. You can create the task flow in the same application that contains the human task or in a separate application.

You must have previously created a human task (`.task` file) as part of a SOA composite before you can create a task flow. See [Chapter 27, "Designing Human Tasks"](#) for how to create the `.task` file.

If the task flow is in the same application as the human task, create a different project for the task flow. If the SOA composite contains multiple human tasks, create a separate project for each ADF task flow associated with each human task. By using an ADF task flow, you create data controls based on the task parameters and outcomes.

To autogenerate an ADF task form, access the human task in the SOA composite application (form and task are in the same application). See [Section 28.3.1, "How To Create an ADF Task Flow from the Human Task Editor,"](#) for more information.

To create an ADF task form in a separate application, create the new application and project and browse for the `.task` file for the human task. See [Section 28.3.2, "How To Create an ADF Task Flow Based on a Human Task,"](#) for more information.

28.3.1 How To Create an ADF Task Flow from the Human Task Editor

The `.task` file that specifies the human task is easily associated with the task flow when the two are located in the same application.

To create an ADF task flow for a human task:

1. Open the BPEL process within the SOA composite application.
2. Double-click the human task activity and click **Edit**.

[Figure 28–4](#) shows the Human Task dialog.

Figure 28–4 *Editing a Human Task*

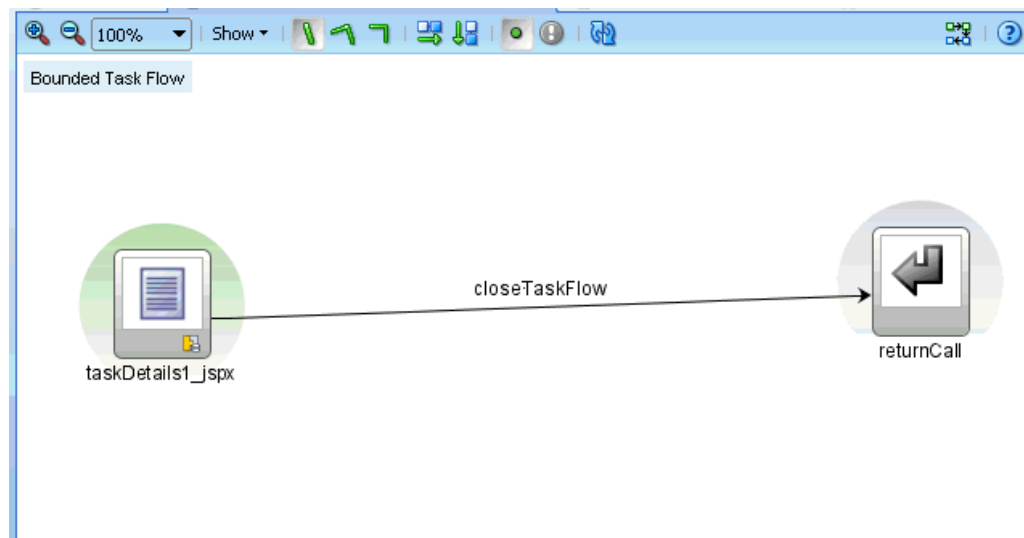
3. In the `.task` tab (shown in [Figure 28–5](#)), click **Create Form** and select **Auto-Generate Task Form**.

Figure 28–5 Creating a Task Flow from the Human Task Editor

The screenshot shows the 'ApprovalHumanTask.task' configuration window. The left sidebar lists categories: General, Data, Assignment, Presentation, Deadlines, Notification, Access, and Events. The 'General' tab is active, displaying the following fields:

- Title:** Plain Text (dropdown), Value: `concat(string('Approval Required for Order Id: '), /task:task/task:payload/task:orderId)`
- Description:** (Empty text box)
- Outcomes:** APPROVE, REJECT
- Priority:** 3 (Normal) (dropdown)
- Category:** By expression (dropdown), Value: (Empty text box)
- Owner:** User (dropdown), Value: weblogic, Static (dropdown)
- Application Context:** (Empty text box)

4. Provide a project name and a directory path (or use the default) and click **OK**.
The `taskDetails1_jsp` icon appears in the designer, as shown in [Figure 28–6](#).

Figure 28–6 The taskDetails1_jsp Icon

The task flow and task form are complete and ready to be deployed.

28.3.2 How To Create an ADF Task Flow Based on a Human Task

The **ADF Task Flow Based on Human Task** option (shown in [Figure 28–1](#)) creates an ADF task flow and additional artifacts to make deployment easier. When you select the `.task` file to associate with the ADF task flow, human task data controls are created based on the task parameters and outcomes. These are then available to use in the JSPX page. You must have access to the SOA composite project while creating the task flow project.

To create an ADF task flow based on a human task:

1. From the **File** main menu, select **New > Applications > SOA Application**.
2. Click **OK**.

3. Provide an application name and directory information (or accept the default), and click **Finish**.
4. Right-click the project name and select **New**.
5. Under **Web Tier**, select **JSF**.
6. Select **ADF Task Flow Based on Human Task** and click **OK**.
7. In the SOA Resource Browser, find and select the `.task` file where you defined the human task and click **OK**.
 - a. If the human task is in the same application as the task definition, then click **File** to use the file browser to navigate to the `.task` file, which is typically in the composite directory.
 - b. If the human task is in a different application, then click **Resource Palette** to use the MDS resource catalog and find the `.task` file in the composite application.
 - c. If the `.task` file is located within the current application, then click **Application**.

This displays the Create Task Flow dialog and creates the data controls.

8. In the Create Task Flow dialog, accept the defaults and click **OK**.

The `taskDetails1_jsp` icon appears in the designer, as shown in [Figure 28–6](#). The task flow has a view, a control flow, and a task return.

To continue creating the task form, see the following:

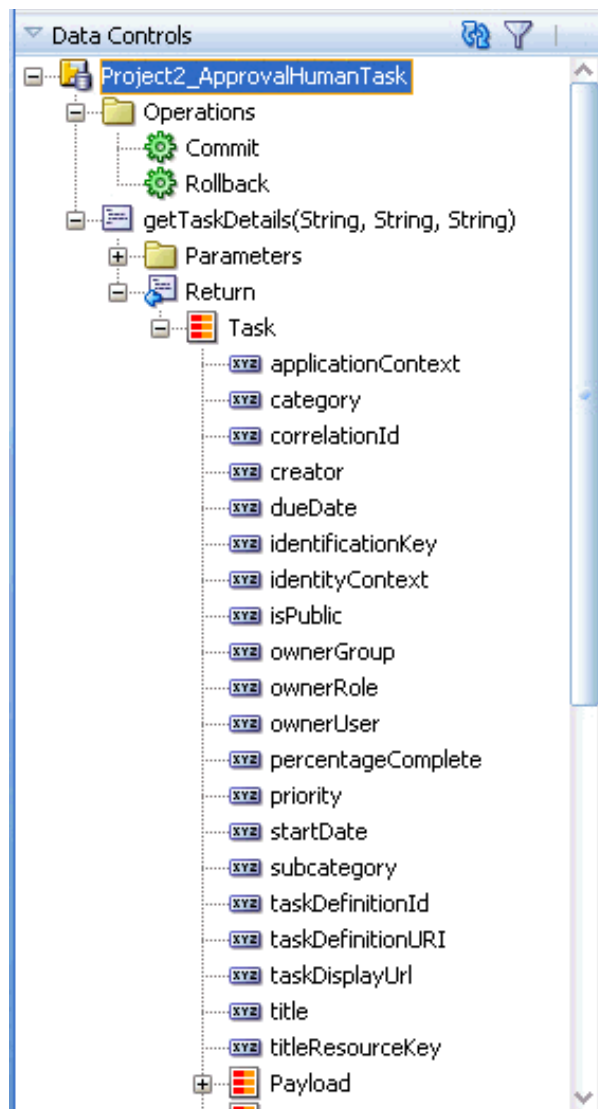
- [Section 28.4.4, "How To Create a Task Form Using the Complete Task with Payload Drop Handler."](#)
- or
- [Section 28.4.5, "How To Create Task Form Regions Using Individual Drop Handlers."](#)

28.3.3 What Happens When You Create an ADF Task Flow Based on a Human Task

With an ADF task flow based on a human task, the task flow application has task data controls that wire the task form with the workflow services. The data controls provide the following:

- Various parameters and operations to access task data and act on it
- Drop handlers with which you can create interface regions to display the contents of the task

The human task-aware data controls appear in the **Data Controls** panel of the Oracle JDeveloper Application Navigator, as shown in [Figure 28–7](#).

Figure 28–7 The Task Collection in the Data Controls Panel

The data controls for the task (represented by the **Task** node in [Figure 28–7](#)) have drop handlers to render the task form. See [Section 28.4, "Creating a Task Form,"](#) for more information.

28.3.4 What You May Need to Know About Having Multiple ADF Task Flows That Contain the Same Element with Different Meta-attributes

You must create separate ADF task flows if both contain the same element, but with different meta-attributes specified (for example, editable and noneditable).

For example, assume you perform the following tasks.

1. Create two task form applications for a SOA composite application:
 - Task form application one (for example, named EnterBankDetails.task) has one editable payload (for example, named BankDetails) and one noneditable payload (for example, named Employee).

- Task form application two (for example, named `ValidatePersonalInformation.task`) has one editable payload (for example, also `Employee`).

While creating the task form, the wizard provides you with the option to define the ADF table for payload `Employee`.

2. Complete the wizard, then deploy the process.
3. Invoke the process.
4. Log in to Oracle BPM Worklist.

There is a `Validate Personal Information` task (for `ValidatePersonalInformation.task`).

5. Select the task.

`Employee` details are available for modification, as expected.

6. Add a new record, then approve the task.
7. Select the `Enter Bank Details` task (for `EnterBankDetails.task`). In the task form, note that the **Insert New** and **Delete** buttons are still present for `Employee` data, even though it is a noneditable payload.
8. Click **Delete**, then select **Approve**. The payload gets deleted.

Ensure that you create two separate ADF task flow applications because both contain the `Employee` element, but with different meta-attributes specified (editable and noneditable).

28.4 Creating a Task Form

You can create a task form by using the **Auto-Generate Task Form** option, the **Launch Task Form Wizard** option, or by using human task drop handlers.

- For how to use the **Auto-Generate Task Form** option, see [Section 28.4.1, "How To Create an Autogenerated Task Form."](#)
- For how to use the **Launch Task Form Wizard** option, see [Section 28.4.3, "How To Create a Task Form Using the Custom Task Form Wizard."](#)
- For how to use human task drop handlers, see the following:
 - [Section 28.4.4, "How To Create a Task Form Using the Complete Task with Payload Drop Handler"](#)
 - [Section 28.4.5, "How To Create Task Form Regions Using Individual Drop Handlers"](#)
 - [Section 28.4.6, "How To Add the Payload to the Task Form"](#)

28.4.1 How To Create an Autogenerated Task Form

The autogenerated task form can be further edited as needed.

To create an autogenerated task form:

1. Open the BPEL process within the SOA composite application.
2. Double-click the human task activity and click **Edit**.
3. From the `.task` editor, click **Create Form** and select **Auto-Generate Task Form**, as shown in [Figure 28–8](#).

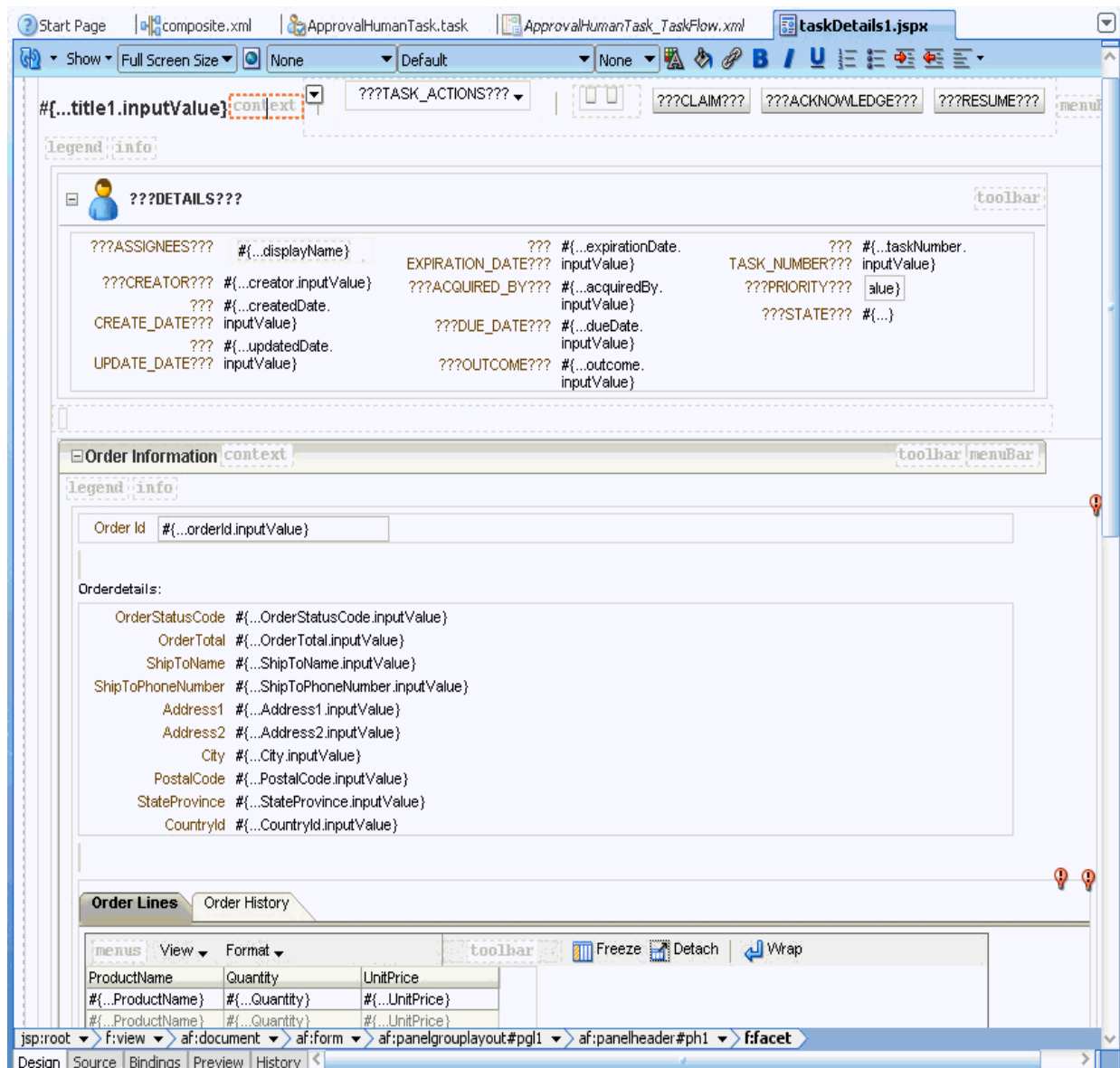
Figure 28–8 Creating a Task Form

The screenshot shows the 'Create Form' dialog for the task 'ApprovalHumanTask.task'. The dialog has a left sidebar with icons for Data, Assignment, Presentation, Deadlines, Notification, Access, and Events. The main area contains the following fields:

- Title:** Plain Text (dropdown) [text box]
- Description:** [text box]
- Outcomes:** [text box]
- Priority:** 3 (Normal) (dropdown)
- Category:** By expression (dropdown) [text box]
- Owner:** User (dropdown) [text box] Static (dropdown)
- Application Context:** [text box]

4. Provide a project name and a directory path (or use the default) and click **OK**.

The default form opens in the **taskDetails1.jspx** tab. The default form for ApprovalHumanTask is shown in [Figure 28–9](#).

Figure 28–9 Autogenerated Task Form for ApprovalHumanTask

28.4.2 How to Register the Library JAR File for Custom Page Templates

You can optionally specify your own custom page templates in the Custom Task Form wizard. As described in [Section 28.4.3, "How To Create a Task Form Using the Custom Task Form Wizard,"](#) you select **Custom Page Template** in the Name and Definition page of the Custom Task Form Wizard and specify the library JAR file name and the path to the .jspx template file within the JAR file.

As a prerequisite, you first must register the library JAR file in Oracle JDeveloper.

To create the library JAR file for custom page templates:

1. From the **Tools** menu, select **Manage Libraries**.
2. Click **New**.

The Create Library dialog appears.

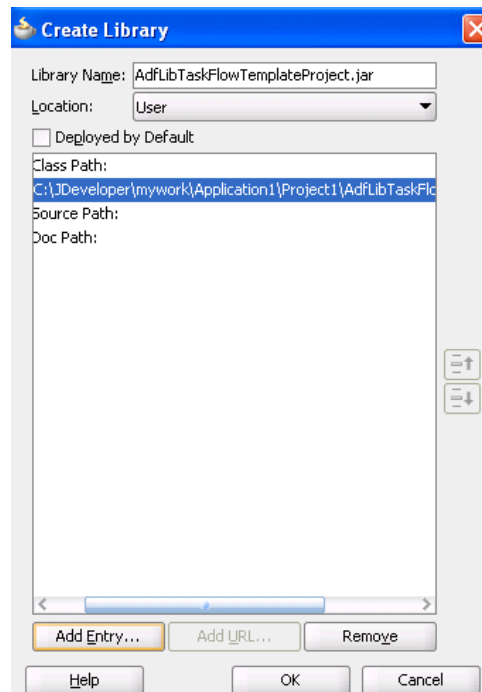
3. Highlight **Class Path**, and click **Add Entry**.

The Select Path Entry dialog appears.

4. Select the class path for the library, and click **Select**.

The class path is displayed below **Class Path** and the library JAR file name is displayed in the **Library Name** field. Ensure that the library name you select ends with a suffix of `.jar`. [Figure 28–10](#) provides details.

Figure 28–10 Custom Library JAR File



5. Select the **Deployed by Default** checkbox.
6. Click **OK**.

When you run the Custom Task Form wizard, you select **Custom Page Template** on the Name and Definition page, and enter the following information that you registered in the Create Library dialog:

- The same library name as you entered in the **Library Name** field in the Create Library dialog; these names must match.
- The path to the `.jspx` templates file in the library JAR file in the **Template Path** field.

28.4.3 How To Create a Task Form Using the Custom Task Form Wizard

This wizard enables you to create a task form using ADF page templates and standardized task regions. The page templates can be either of the following:

- Default page templates that are automatically provided at the following location:
`[JDeveloper_Home]\jdeveloper\soa\modules\oracle.soa.worklist_11.1.1\adfLibWorklistComponents.jar`

The default page templates are:

- Nontabbed, default template: `taskDetailsTemplate.jspx`
- Tabbed templates in which the payload and comments, attachment, and history sections are displayed on a separate tab:
`taskDetailsTemplate2.jspx`

You select **Default Page Template** in the Name and Definition page of the Custom Task Flow wizard.

- Custom page templates that you define. You select **Custom Page Template** in the Name and Definition page of the Custom Task Flow wizard. You package a page template and its artifacts into an ADF library JAR file. These JAR files can be packaged, deployed, discovered, and used like any other Oracle library component. The wizard prompts you to specify the JAR name and template location in the JAR.

Page templates let you define entire page layouts, including values for certain attributes of the page. When pages are created using a template, they all inherit the defined layout. When you make layout modifications to the template, all pages that consume the template automatically reflect the layout changes.

The templates used in the wizard generate content for the following six facets:

- Actions
- Attachments
- Body
- Comments
- Header
- History

For the action, header, and body facets, you can pick the content and attributes that you want to display and then fine tune the layout.

All six facets are defined in the default page templates. In the case of custom templates, you use these exact facet names in your template. If your template does not include these facets, then the facet content is not generated in the JSPX file.

If you do not want to include page templates in your task form, select **None** in the Name and Definition page. In this case, the wizard generates a task form with a header, body (one or more), and footer, and provides for tabular formatting into columns and rows. You can select any of the task (system) actions to display on the form and you can specify that the custom actions defined for the human task appear on the form as buttons. Any or all parts of the payload can be selected to appear, as well as attachments and comments.

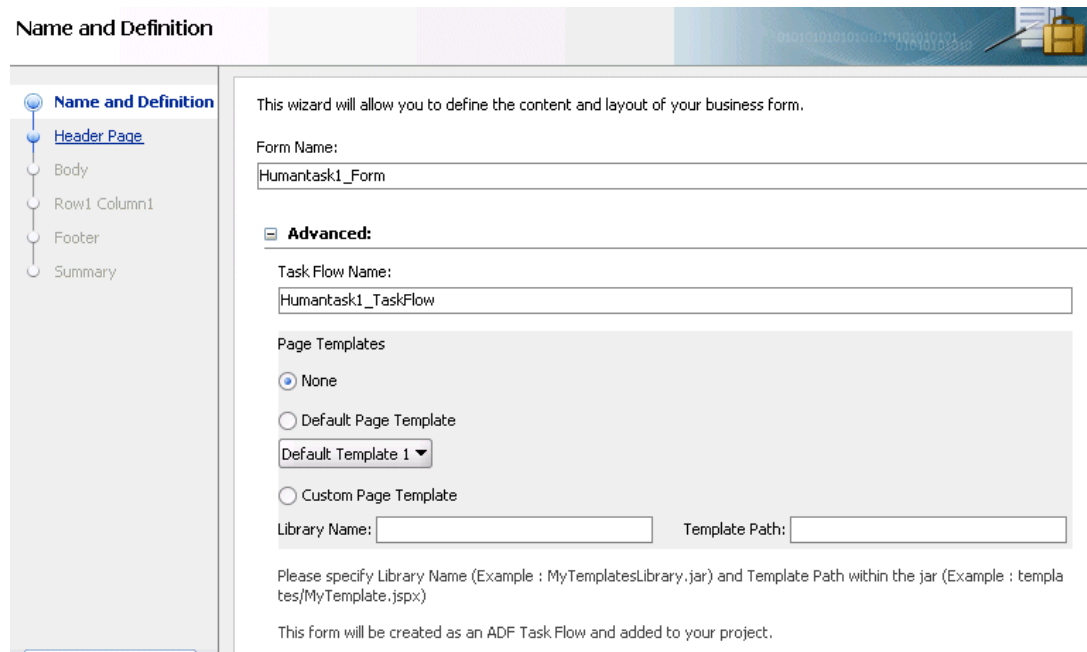
For more information about facets, see *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

To create a custom task form:

1. Open the BPEL process within the SOA composite application.
2. Double-click the human task activity, and click the **Edit** icon.
The Human Task Editor appears.
3. Above the editor, click **Create Form** and select **Launch Task Form Wizard**.
4. Provide a project name and a directory path (or use the default), and click **OK**.

5. In the **Form Name** field on the Name and Definition page, shown in [Figure 28–11](#), provide the name of the form (.jspx file) that is generated at the end of the wizard. The default name, `Humantasknumber_Form`, is provided if you do not provide a name. Ensure that valid characters are used in the name. Spaces are not permitted.

Figure 28–11 Custom Task Form Wizard: Form Name and Definition



Name and Definition

This wizard will allow you to define the content and layout of your business form.

Form Name:
Humantask1_Form

Advanced:

Task Flow Name:
Humantask1_TaskFlow

Page Templates

☒ None

☐ Default Page Template

Default Template 1 ▼

☐ Custom Page Template

Library Name: Template Path:

Please specify Library Name (Example : MyTemplatesLibrary.jar) and Template Path within the jar (Example : templates/MyTemplate.jsxpx)

This form will be created as an ADF Task Flow and added to your project.

6. If you want to specify advanced definitions, expand **Advanced**.
7. Specify the following information, and click **Next**. This form is created as an ADF task flow and added to the project.
 - **Task Flow Name:** The name of the ADF task flow that is generated at the end of the wizard. Accept the default name of `Humantasknumber_TaskFlow` or specify a different name.
 - **Page Templates:** The page template to use.
 - **None:** Select to create a task form that does not use a page template. This is the default selection.
 - **Default Page Template:** Select to use the default page template.
 - **Custom Page Template:** Specify the library name (for example, `MyTemplatesLibrary.jar`) and template class path in the JAR file (for example, `templates/MyTemplate.jsxpx`) to use. This is the library name and template class path that you specified in the Create Library dialog in [Section 28.4.2, "How to Register the Library JAR File for Custom Page Templates."](#)

If the template is not found after the wizard finishes, a message is displayed indicating that the template was not found, and that task form generation is to continue without the use of a template.
8. On the Header page, shown in [Figure 28–12](#), perform the following procedures and click **Next**.

- In the **Actions facet** section, select the options to include in the title bar of the task form:
 - **Other actions (menu)**: Lists the system actions that are possible for the task, such as **Request Information**, **Reassign**, **Renew**, **Suspend**, **Escalate**, and **Save**.
 - **Outcomes (buttons)**: Displays buttons for task actions that are defined in the human task, such as setting task outcomes.
- In the **Header facet** section, enter the number of display columns. If you want each header label to display in its own column, then enter the same number as the number of headers you move into the **Selected** list. If you enter 1, but select 7 headers, all 7 headers appear in one column.
- Move header labels into the **Selected** list and reorder them as needed.

Figure 28–12 Custom Task Form Wizard: Setting Up the Header

The screenshot shows the 'Header Page' of the Custom Task Form Wizard. On the left is a navigation pane with links: 'Name and Definition', 'Header Page' (selected), 'Body', 'Row1 Column1', 'Footer', and 'Summary'. The main area is titled 'Header Page' and contains two sections: 'Include in title bar:' and 'Select Header contents and order.'.

The 'Include in title bar:' section has a sub-section 'Actions facet' with two checked options: 'Other actions (menu)' and 'Outcomes (buttons)'.

The 'Select Header contents and order.' section has a sub-section 'Header facet'. It includes a 'Display in' dropdown set to '1' and the text 'columns'. Below this are two lists: 'Available' and 'Selected'. The 'Available' list contains: 'Acquired By', 'Assignees', 'Created Date', 'Creator', 'Due Date', 'Expiration Date', 'Outcome', 'Priority', 'State', 'Task Number', and 'Updated Date'. The 'Selected' list is currently empty. Between the lists are four arrow buttons: a single right arrow, a double right arrow, a single left arrow, and a double left arrow. To the right of the 'Selected' list are four arrow buttons: a single up arrow, a double up arrow, a single down arrow, and a double down arrow.

9. On the Body page, shown in [Figure 28–13](#), perform the following procedures in the **Body facet** section to set up the form, and click **Next**:
 - Enter a title that describes the body panel.
 - Enter the number of columns for row 1. For a simple form, you may want to enter the same number as you entered for the number of header columns.
 - Click the **Add (+)** button to add more rows. For each new row, you can also specify the number of columns. Each row can have its own column layout. For each column in each row, a body page is created, labeled Row1, Column1, and so on.

Figure 28–13 Custom Task Form Wizard: Setting Up the Body

The screenshot shows the 'Body Page' of the Custom Task Form Wizard. On the left is a navigation pane with a tree view containing: Name and Definition, Header Page, Body (selected), Row1 Column1, Footer, and Summary. The main area is titled 'Body facet' and contains a 'Panel title:' text box. Below this is the instruction 'Specify the number of rows and columns for the body of your form'. There are two input fields: 'Number of Columns' with a spinner set to '1', and 'Add/Delete Rows' with a '+' button.

Note: If you specify rows or columns for which no payload data appears, then an empty panel group is displayed. You can use this blank section to add content to the form later by using data controls.

10. On the Row1 Column1 page, shown in [Figure 28–14](#), move all or part of the payload to the **Selected** list and click **Next**.

Figure 28–14 Custom Task Form Wizard: Selecting the Body Fields

The screenshot shows the 'Row1 Column1' page of the Custom Task Form Wizard. The navigation pane on the left has 'Row1 Column1' selected. The main area is titled 'Select content to be added to the indicated body section' with a checkbox. It features two list boxes: 'Available' on the left containing 'Payload' and 'orderId', and 'Selected' on the right containing 'Payload'. Between the lists are four arrow buttons for moving items: a single right arrow, a double right arrow, a single left arrow, and a double left arrow.

11. For any Row*n* Column*n* page after Row1 Column1, repeat Step 10 and click **Next**.

The Footer page that displays is based upon the page template you selected on the Name and Definition page in Step 7 (either Default Page Template or Custom Page Template).

- a. If you selected Default Page Template, the Footer page shown in [Figure 28–15](#) is displayed. Deselect any comments, attachments, or history facet that you do not want to include in the footer, and click **Next**. By default, the comments, attachments, and history facets are all selected.

Figure 28–15 Custom Task Form Wizard: Selecting the Footer Fields for the Default Page Template

- b. If you selected Custom Page Template, the Footer page shown in [Figure 28–16](#) is displayed. Select any comments, attachments, or history facet that you want to move to the **Selected** list, and click **Next**.

Figure 28–16 Custom Task Form Wizard: Selecting the Footer Fields for the Custom Page Template

12. On the Summary page, shown in [Figure 28–17](#), inspect your selections. Click **Back** to make changes or click **Finish**.

Figure 28–17 Custom Task Form Wizard: Summary

Summary

Navigation Tree:

- Name and Definition
- Header Page
- Body
- Row1 Column1
- Row1 Column2
- Footer
- Summary**

Header

- Acquired By
- Assignees
- Created Date
- Creator
- Due Date
- Expiration Date
- Outcome
- Priority
- State
- Task Number
- Updated Date

Body

_ApprovalHumanTask.payloadType.orderId	

Footer

- Attachments
- Comments
- History

The Designer initializes and the `form_name.jspx` tab is displayed, as shown in [Figure 28–18](#) (upper part of tab) and [Figure 28–19](#) (lower part of tab).

Figure 28–18 Custom Task Form (Upper Part of Tab)

Actions: [Dropdown] [Claim] [Dismiss] [Resume]

#...title.inputValue

Assignees: #{...displayName}

Acquired By: #{...acquiredBy.inputValue}

Created: #{...createdDate.inputValue}

Creator: #{...creator.inputValue}

Due Date: #{...dueDate.inputValue}

Expiration Date: #{...expirationDate.inputValue}

Outcome: #{...actionDisplayName.inputValue}

Priority: [value]

State: #{...}

Task Number: #{...taskNumber.inputValue}

Updated: #{...updatedDate.inputValue}

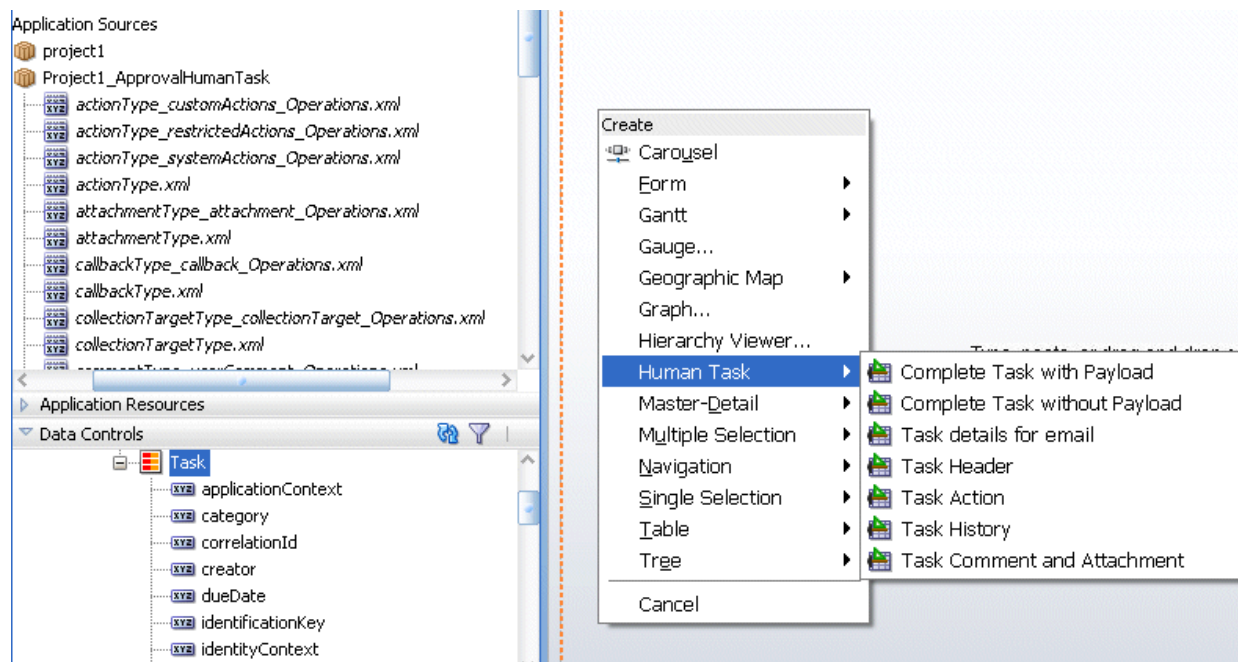
Figure 28–19 Custom Task Form (Lower Part of Tab)

The screenshot shows the lower part of a custom task form with three main sections, each with a 'context' menu and 'legend'/'info' tabs.

- Comments context:** Contains a list of comments with fields for `#{...updatedDate}` and `#{...updatedBy.id}`.
- Attachments context:** Contains a list of attachments with the field `#{...name}`.
- History context:** Contains a table with columns for task snapshots and future participants. The table has headers like `TABLE_COL_PARTIC`, `TABLE_COL_ACTION`, `TABLE_COL_UPDATEDBY`, and `TABLE_COL_ACTION_DATE`.

28.4.4 How To Create a Task Form Using the Complete Task with Payload Drop Handler

The human task drop handlers appear in the context menu of the designer, as shown in Figure 28–20.

Figure 28–20 Human Task Drop Handlers for Creating the Task Form

Other ADF drop handlers—for forms, tables, trees, and so on (shown in [Figure 28-20](#))—can also be used to create task forms. See *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* for more information about standard ADF drop handlers.

Complete Task with Payload

This option creates the combination of all the preceding task form components (the task header, task history, task actions, and task comments and attachments), plus the interface for the payload. The payload interface is created as follows:

- All text nodes are created as text input fields.
- If an element has `maxOccurs="unbounded"`, then it appears as a table.
- Nested tables are not rendered; that is, if an element has `maxOccurs="unbounded"` and it has a child with `maxOccurs="unbounded"`, then the child element is not rendered.
- If there are multiple levels of nesting, then drag and drop the individual sections and use a standard ADF drop handler.

Complete Task without Payload

This option creates the combination of all of the preceding task form components (the task header, task history, task actions, and task comments and attachments).

Task Details for Email

This option creates an ADF region that renders well when sent by email. It generates the form shown in [Figure 28-21](#).

Figure 28–21 Task Form for Email Notification

Task Number: #{...taskNumber.inputValue}	Creator: #{...creator.inputValue}	Assignees: #{...id}[#{...}]
State: #{...}	Created Date: #{...createdDate.inputValue}	Acquired By: #{...acquiredBy.inputValue}
Outcome: #{...outcome.inputValue}	Updated Date: #{...updatedDate.inputValue}	
Priority: low	Expiration Date: #{...expirationDate.inputValue}	

#{...hints.label}	#{...Location.inputValue}
#{...hints.label}	#{...type.inputValue}
#{...hints.label}	#{...problemDescription.inputValue}
#{...hints.label}	#{...severity.inputValue}
#{...hints.label}	#{...status.inputValue}

#{...hints.label}	#{...ID.inputValue}
#{...hints.label}	#{...FirstName.inputValue}
#{...hints.label}	#{...LastName.inputValue}
#{...hints.label}	#{...Email.inputValue}
#{...hints.label}	#{...phone.inputValue}

See [Section 28.7, "Creating an Email Notification,"](#) for more information.

Task Header

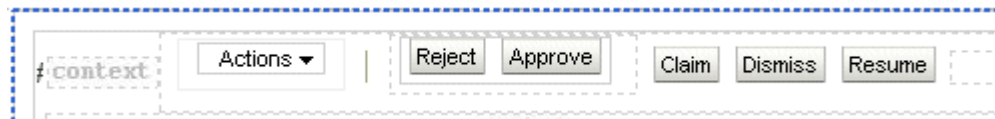
All the standard header fields are added to the task form. This includes the task number and title; the state, outcome, and priority of the BPEL process, and information about who created, updated, claimed, or is assigned to the task. The header also displays dates related to task creation, last modification, and expiration. You can add or remove header fields as required for your task display.

[Figure 28–22](#) shows an example of header information.

Figure 28–22 Header Information

 Details toolbar		
Assignees: #{...displayName}	Expiration Date: #{...expirationDate.inputValue}	Task Number: #{...taskNumber.inputValue}
Creator: #{...creator.inputValue}	Acquired By: #{...acquiredBy.inputValue}	Priority: low
Created: #{...createdDate.inputValue}	Due Date: #{...dueDate.inputValue}	State: #{...}
Updated: #{...updatedDate.inputValue}	Outcome: #{...outcome.inputValue}	

Buttons for task actions are also created in the header, as shown in [Figure 28–23](#).

Figure 28–23 Task Header: Task Action Buttons

Task Actions

The following task actions appear from the **Actions** dropdown list or as buttons. The tasks that appear depend on the state of the task (assigned, completed, and so on) and the privileges that are granted to the user viewing the task. For example, when a task is assigned to a group, only the **Claim** button appears. After the task is claimed, other actions such as **Reject** and **Approve** appear.

The first three custom actions appear on the task form as buttons: **Claim**, **Dismiss**, and **Resume**. Only those buttons applicable to the task appear. Other actions are displayed under the **Actions** list, starting with **Request for Information**, **Reassign**, and **Route**. Systems actions—**Withdraw**, **Pushback**, **Escalate**, **Release**, **Suspend**, and **Renew**—follow the custom actions, followed by the **Save** button. These actions require no further dialog to complete.

- **Claim**—A task that is assigned to a group or multiple users must be claimed first. **Claim** is the only action available in the **Task Action** list for group or multiuser assignments. After a task is claimed, all applicable actions are listed.

Note: If an FYI task is sent to multiple users, a user must first select the **Claim** button to claim the task before they can dismiss it.

- **Dismiss**—This action is used for a task that requires the person acting on the task to acknowledge receipt, but not take any action (like an FYI).
- **Resume**—A task that was halted by a **Suspend** action can be worked on again. See **Suspend**.
- **Request for Information**—You can request more information from the task creator or any of the previous assignees. If reapproval is not required, then the task is assigned to the next approver or the next step in the business process.
- **Reassign**—Managers can reassign a task to reportees. The **Reassign** option also provides a **Delegate** function. A task can be delegated to another user or group. The delegated task appears in both the original user's and the delegated user's worklists. The delegated user can act on behalf of the original assignee, and has the same privileges for that task as the original assignee.
- **Route**—If there is no predetermined sequence of approvers or if the workflow was designed to permit ad hoc routing, then the task can be routed in an ad hoc fashion. For such tasks, a **Route** button appears on the task details page. From the Routing page, you can look up one or more users for routing. When you specify multiple assignees, you can choose whether the list of assignees is for simple (group assignment to all users), sequential, or parallel assignment. In the case of parallel assignment, you provide the percentage of votes required for approval.
- **Withdraw**—Only the task creator can withdraw (cancel) the task. The **Comments** area is available for an optional comment. The business process determines what happens next.
- **Pushback**—This action sends a task up one level in the workflow to the previous assignee.

- **Escalate**—An escalated task is assigned to the user's manager. The **Comments** area is available for an optional comment.
- **Release**—Releasing a task makes it available to other assignees. A task assigned to a group or multiple users can then be claimed by the other assignees.
- **Suspend**—This action suspends the expiration date indefinitely, until the task is resumed. Suspending and resuming tasks are available only to users who have been granted the `BPMWorkflowSuspend` role. Other users can access the task by selecting **Previous** in the task filter or by looking up tasks in the Suspended status. Buttons that update a task are disabled after suspension.
- **Renew**—Renewing a task extends the task expiration date seven days (P7D is the default). The renewal duration is controlled from Oracle Enterprise Manager Grid Control Console. A renewal appears in the task history. The **Comments** area is available for an optional comment.
- **Save**—Changes to the task are saved.

While you are creating a task form, all possible system action buttons appear, although only those actions that are appropriate for the task state and fit the user's privileges appear in the worklist.

Task History

The history of task actions appears on the task details page, and is displayed in the worklist as a history table. The history includes the following fields:

- Version number
- Participant name—the person who acted on the task
- Action—for example, if the task was approved or assigned
- Updated By—name of the person who last updated the task
- Action date

See [Figure 29–20, "History: Graphical View"](#) and [Figure 29–21, "History: Full Task Actions"](#) for how task history is displayed in Oracle BPM Worklist, including the options to take a history snapshot, list future participants, and list full task actions.

Task Comments and Attachments

A trail of comments with the comment date and comment writer's user name is maintained throughout the life cycle of a task.

Files or reference URLs associated with a task can be added by any of the human task participants.

[Figure 28–24](#) shows an example of the comments and attachments region.

Figure 28–24 Comments and Attachment Region

The screenshot displays two side-by-side panels from the Oracle BPM Worklist interface. The left panel is titled 'Comments' and contains a table with columns 'legend' and 'info'. The table has a header row with '#{...updatedDate}' and '#{...updatedBy.id}'. Below the header, there are four rows of data, each containing a comment text and the user who updated it. The right panel is titled 'Attachments' and also has a 'legend' and 'info' column structure. It shows a table with a header row containing '#{...name}' and a single row of data below it.

The following steps describe how to use a drop handler that creates the task form, including the payload, without building each region individually. To build each region individually, see [Section 28.4.5, "How To Create Task Form Regions Using Individual Drop Handlers."](#)

To create a task form using the Complete Task with Payload drop handler:

1. In the designer, double-click `taskDetails1.jspx`.
2. In the Create JSF Page dialog, provide a file name and directory information (or accept the defaults) and click **OK**.
3. In the **Data Controls** panel of the Application Navigator, expand the human task node, then the `getTaskDetails` node, and then the **Return** node.
4. Drag the **Task** icon into the `taskDetails.jspx` window.
5. Select **Human Task**, and then **Complete Task with Payload**.
6. In the Edit Action Binding dialog, shown in [Figure 28–25](#), click **OK**.

Figure 28–25 *Edit the Action Binding*

Select a data collection and the action you want your control to initiate. The control initiates the action on the data objects of the selected collection.

Data Collection:

- Project1_ApprovalHumanTask

Select an Iterator: New...

Operation: `getTaskDetails(String, String, String)`

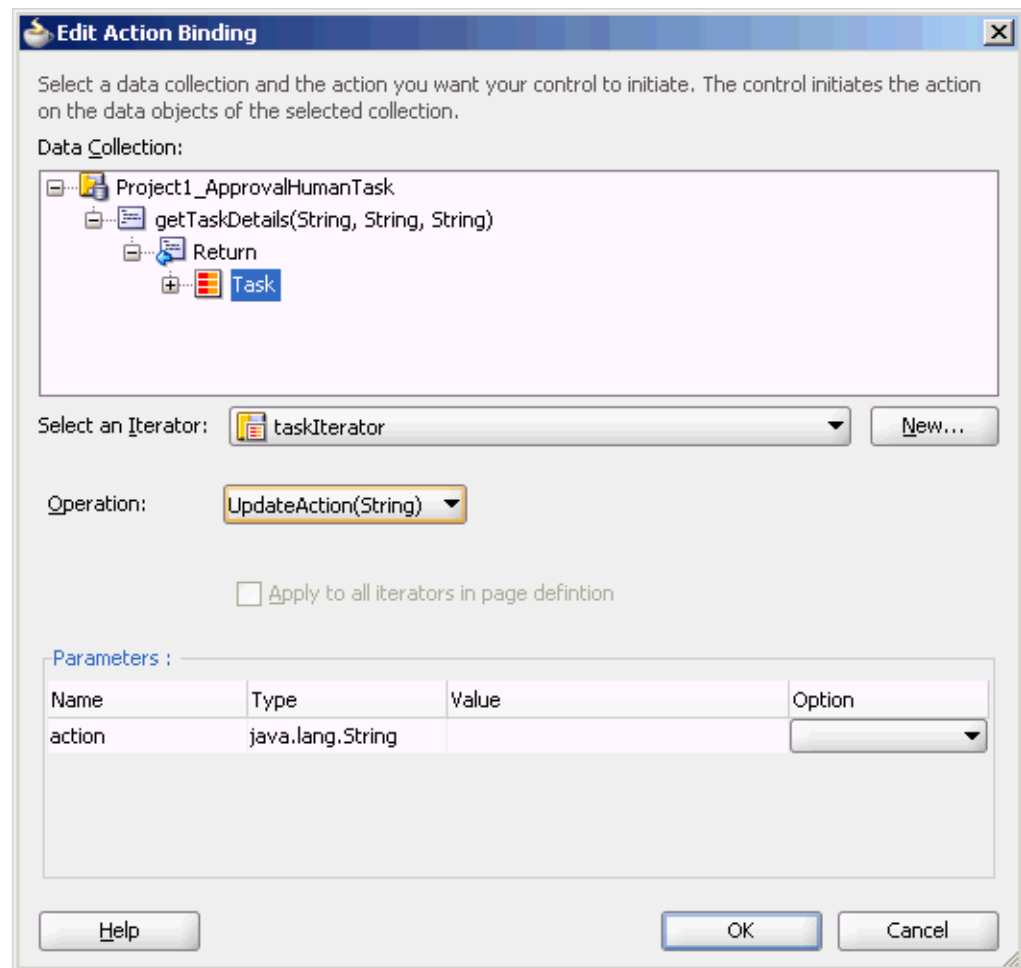
☐ Apply to all iterators in page definition

Parameters :

Name	Type	Value	Option
bpmWorklistContext	java.lang.String		▼
bpmWorklistTaskId	java.lang.String		▼
bpmWorklistTaskV...	java.lang.String		▼

Help OK Cancel

7. In the next Edit Action Binding dialog, the data collection is selected, as shown in [Figure 28–26](#); click **OK**.

Figure 28–26 *Select the Data Collection and Action*

The task form is displayed, as shown in [Figure 28–27](#).

Figure 28–27 Task Form

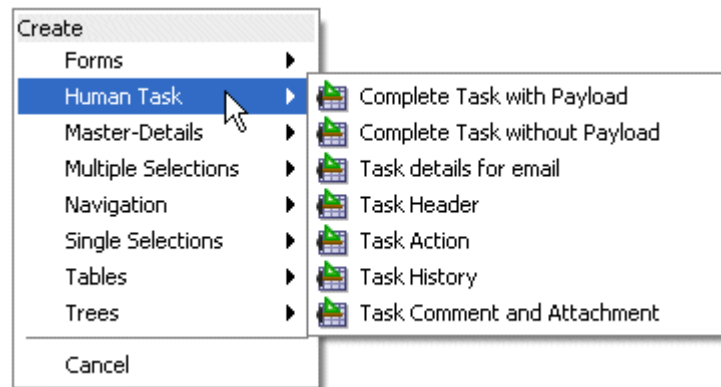
The screenshot displays the Oracle Fusion Middleware Task Form Designer interface. The top toolbar includes buttons for 'Claim', 'Dismiss', and 'Resume'. The main form area is divided into several sections:

- Assignees:** A section with input fields for 'Assignees', 'Creator', 'Created', 'Updated', 'Expiration Date', 'Acquired By', 'Due Date', 'Outcome', 'Task Number', 'Priority', and 'State'.
- Order Information:** A section with a 'Panel Group Layout - vertical' containing input fields for 'Order Id', 'OrderStatus', 'OrderTotal', 'ShipToName', 'ShipToPhoneNumber', 'Address1', 'Address2', 'City', 'PostalCode', 'StateProvince', and 'CountryId'.
- Order Lines:** A section with a table showing 'Product Name', 'Quantity', and 'Unit Price'.

The bottom of the form features a 'Click to set cursor after Spacer - 20' button and a 'Click to set cursor after Spacer - 20' button.

28.4.5 How To Create Task Form Regions Using Individual Drop Handlers

You can create a display form with multiple regions using the individual **Task Header**, **Task Action**, **Task History**, and **Task Comment and Attachment** drop handlers shown in Figure 28–28.

Figure 28–28 Using Human Task Drop Handlers

Task Header provides both header and task actions, so you do not need the **Task Action** drop handler when you use **Task Header**. Use **Task Action** when you want the actions dropdown menu and buttons, but not header details.

To create the task form without building each region individually, see [Section 28.4.4, "How To Create a Task Form Using the Complete Task with Payload Drop Handler."](#)

Before you create this task form, you must have created the following:

- A new application and SOA project, and a human task service.
- An ADF task flow based on the human task. See [Section 26.3.2.2, "How to Create the Vacation Request Process,"](#) for more information.

To create task form regions using individual drop handlers:

1. In the designer, double-click `taskDetails1.jspx`.
2. In the Create JSF Page dialog, provide a file name and directory information (or accept the defaults) and click **OK**.
3. In the **Data Controls** panel of the Application Navigator, expand the human task node, then the `getTaskDetails` node, and then the **Return** node.
4. Drag the **Task** icon into the `taskDetails1.jspx` window.
5. Select **Human Task**, and then **Task Header**.

This creates the **Actions** dropdown list and buttons for task actions, as shown in [Figure 28–29](#), and header details, as shown in [Figure 28–30](#).

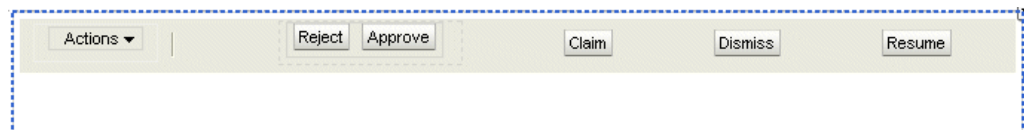
Figure 28–29 Designing the Task Form: Buttons for Task Actions

Figure 28–30 Designing the Task Form: Task Headers

The screenshot shows a 'Details' panel in an ADF Faces application. It contains several input fields with EL expressions for their values:

Assignees	<code># {...displayName}</code>	Expiration Date	<code># {...expirationDate.inputValue}</code>	Task Number	<code># {...taskNumber.inputValue}</code>
Creator	<code># {...creator.inputValue}</code>	Acquired By	<code># {...acquiredBy.inputValue}</code>	Priority	<code>value</code>
Created	<code># {...createdDate.inputValue}</code>	Due Date	<code># {...dueDate.inputValue}</code>	State	<code># {...}</code>
Updated	<code># {...updatedDate.inputValue}</code>	Outcome	<code># {...outcome.inputValue}</code>		

6. Drag additional **Task** icons into the JSPX designer, selecting these options with each iteration:

- **Human Task**, then **Task History**
- **Human Task**, then **Task Comment and Attachment**

The task form now has multiple regions for task action dropdown lists and buttons, task header details, task history, and comments and attachments.

To continue creating the task form, see Step 1 in [Section 28.4.6, "How To Add the Payload to the Task Form."](#)

28.4.6 How To Add the Payload to the Task Form

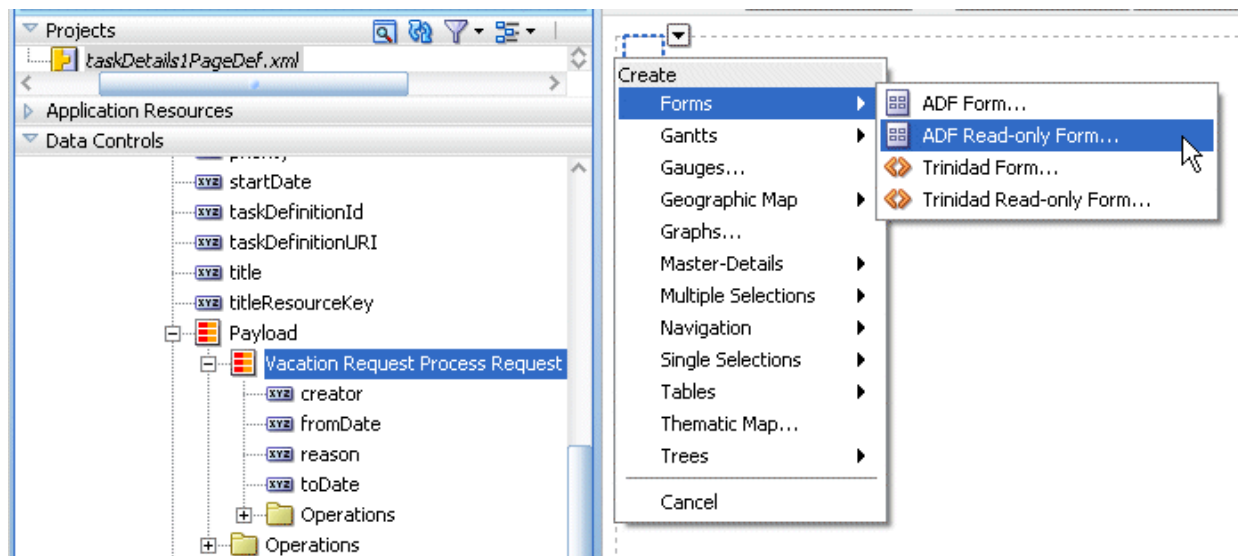
In addition to adding the payload, you can create task form regions. See Step 1 in [Section 28.4.5, "How To Create Task Form Regions Using Individual Drop Handlers."](#)

To add the payload to the task form:

1. From the **Component Palette**, select **ADF Faces**.
2. Expand **Layout**.
3. Drag **Panel Group Layout** between the **Header** and **Comment** sections.
4. In the **Data Controls** panel, expand **Task**, and then **Payload**.
5. Drag the payload data collection to the left of the **Panel Group Layout** area.

An alternative to dropping the payload node onto the form is to expand the payload node and drop the necessary child elements onto the form. For example, to create a read-only form for the VacationRequest payload, expand the payload node, drag the Vacation Request Process Request node onto the form, and select **Forms > ADF Read-only Form**.

6. From the context menu, select **Forms**, then **ADF Read-only Form**, as shown in [Figure 28–31](#).

Figure 28–31 Adding ADF Read-Only Fields to the Task Form Payload Region

7. In the Edit Form Fields dialog, accept the defaults and click OK.

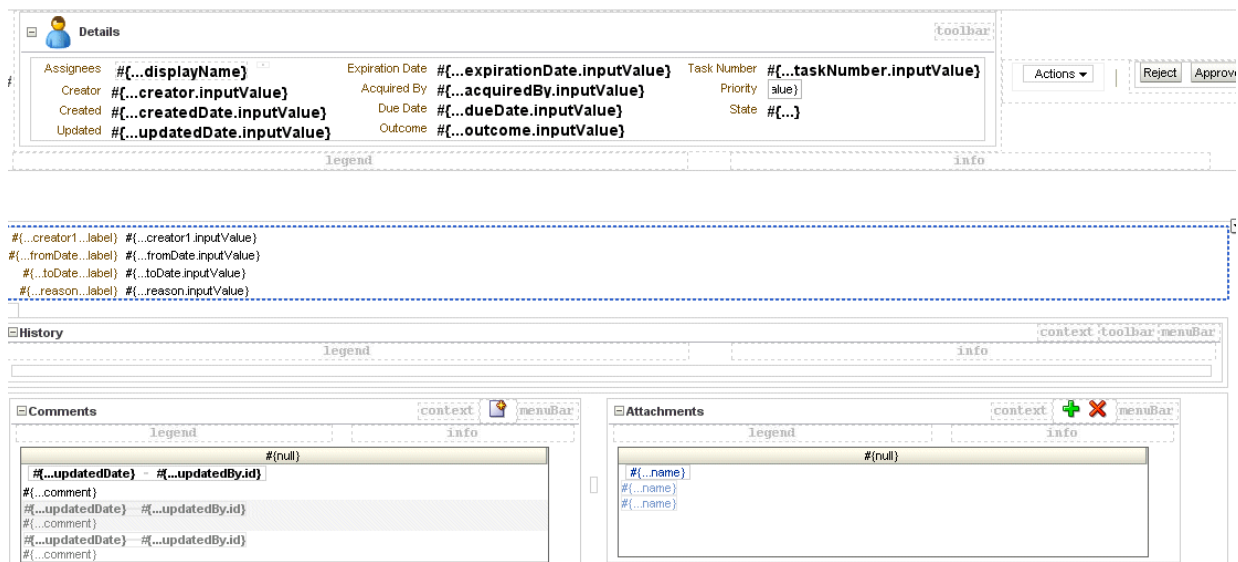
This creates read-only fields in the payload region, between the **Details** and **History** sections.

The payload regions appear, as shown in [Figure 28–32](#).

Figure 28–32 The Payload Region of the Task Form

```
#{...creator1...label} #{...creator1.inputValue}
#{...fromDate...label} #{...fromDate.inputValue}
#{...toDate...label} #{...toDate.inputValue}
#{...reason...label} #{...reason.inputValue}
```

The task form, shown in [Figure 28–33](#), is complete and ready to be deployed.

Figure 28–33 The Task Form (taskDetails.jspx)

28.4.7 What Happens When You Create a Task Form

The form you designed is saved in the .jspx file at

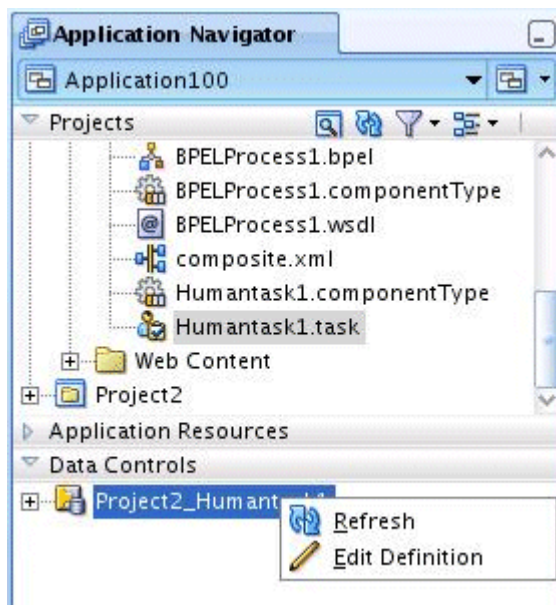
`JDev_Oracle_Home\mywork\task_form_application_name\project_name\public_html`

The task form is ready to be deployed. See [Section 28.8, "Deploying a Composite Application with a Task Flow."](#)

28.5 Refreshing Data Controls When the Task XSD Changes

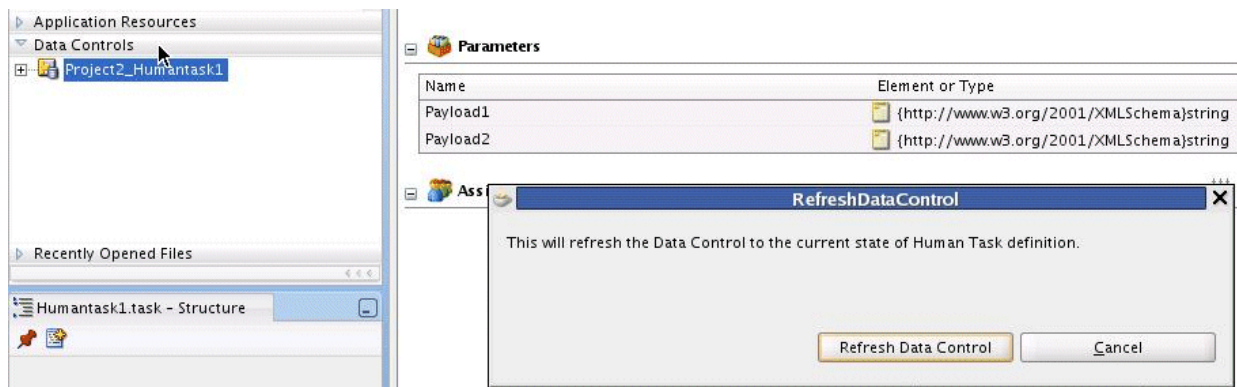
When task metadata changes, refresh the *Data Controls view* (XSD changes are not refreshed) that is based on that task metadata. The refresh functionality re-creates the data control. [Figure 28–34](#) shows the **Refresh** option.

Figure 28–34 Refreshing Data Controls



To refresh the data control:

1. Right-click the data control.
2. Select the **Edit Definition** option to display the Refresh Data Control dialog, as shown in [Figure 28–35](#).

Figure 28–35 The Refresh Data Control Button

28.6 Securing the Task Flow Application

You can use any container-based security for securing the task flow. See [Section 31.6.2.1.2, "Requirements for Client Applications For Identity Propagation,"](#) for more information. Form-based authentication and SSO-based authentication are available for web security.

If you are sending a notification as email, do not secure the URL with `/notification/secure` to use container-based security because this is accessed by SOA APIs using an internal context that cannot be created outside of SOA. The URL pattern inside security cannot contain `/` (all URLs) and `/notification`.

No additional steps are required for identity propagation. Identity is automatically propagated to the server EJBs.

28.7 Creating an Email Notification

A task form is used to provide an email notification, if email notification is defined as part of the human task. Options for email notification include:

- **Default email notification**—Use the first page of the task form that you create for the human task. The content is sent as an HTML-based email. Images in the task flow are included as attachments so that the notification can be viewed in disconnected mode. All drop handlers, including **Complete Task with Payload** and **Complete Task without Payload**, are suitable for emails.
- **Custom email notification**—Use the **Task display for email** drop handler to create a custom email notification task page.

[Section 31.2, "Notifications from Human Workflow"](#) to review notification settings as part of a human task definition (`.task` file).

28.7.1 How To Create an Email Notification

To send a custom email notification whose content and layout you have specified, create another JSPX file in which you design an email notification page. (Note, however, that you can use the default page for notification with no further modifications.) Create the custom notification page by using the custom and standard drop handlers, or use the email notification drop handler. In addition, do the following:

- Add a router to the task flow. The router directs the task flow to send either the email notification page or the default page, depending on the control flow based on the `bpmClientType` page flow scope value.
- Edit the generated inline CSS to customize the page. No additional CSS is included at runtime and the ADF CSS is not available at runtime. See the samples `notification-101` and `notification-102` at <https://soasamples.samplecode.oracle.com>
- 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.)

28.7.1.1 Creating a Task Flow with a Router

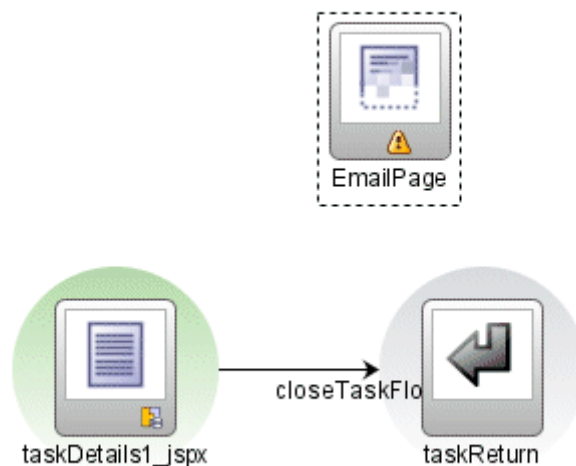
The control flow case with a router enables you to direct the request to a specific page based on certain parameters. For an ADF task flow based on a human task, you need a special page for email notifications. This section describes how to create a special page for email notifications.

To create a task flow with a router:

1. In the Application Navigator, expand the task flow project and double-click *project_name_TaskFlow.xml*.
The XML file opens in the designer. In the diagram view, you see the **taskDetails1.jspx** icon.
2. From the **Component Palette**, select **ADF Task Flow**, and drag the **View** icon into the designer.
3. Click **view1** below the icon and enter a name for the email notification page.

Figure 28–36 shows an example using the name **EmailPage**.

Figure 28–36 Creating the Email Page



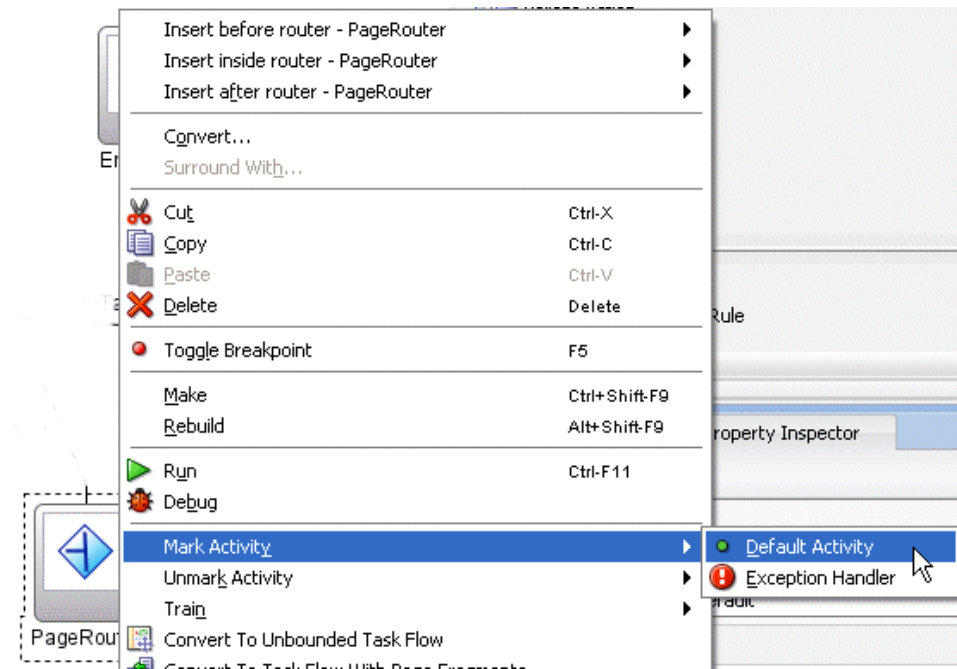
4. From the **Component Palette**, drag **Router** into the designer.

5. Click **router1** below the icon and enter a router name.

Figure 28–38 shows an example using the name **PageRouter**.

6. To ensure that the router is called, right-click the router icon and click **Mark Activity > Default Activity**, as shown in Figure 28–37.

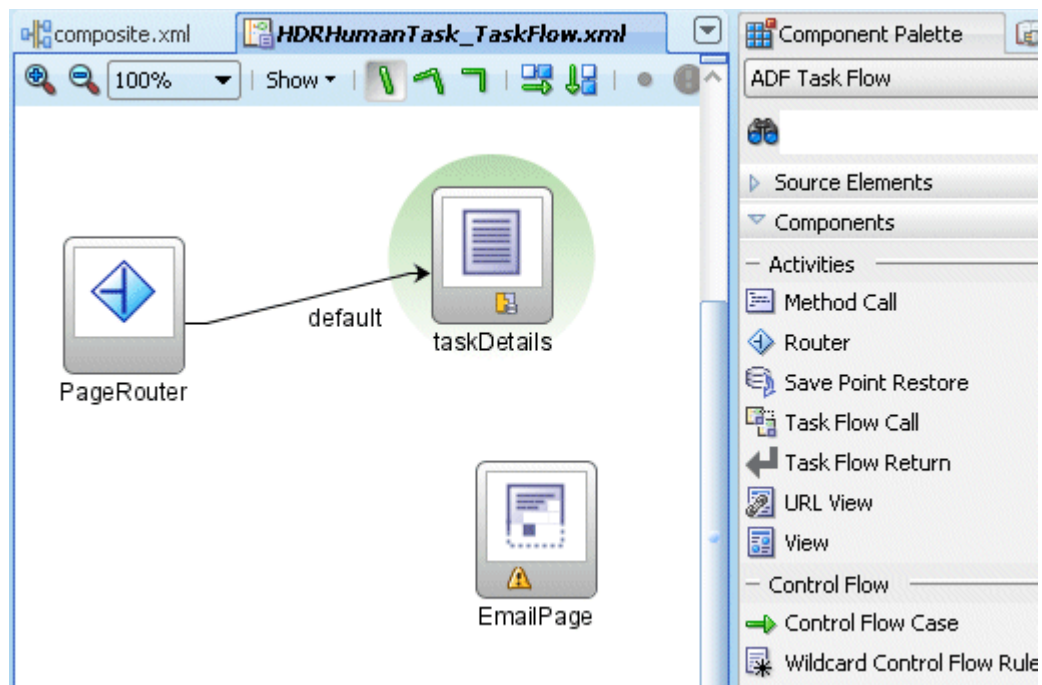
Figure 28–37 Marking the Router as the Default Activity



7. Click the **router - router_name - Property Inspector** tab.
8. In the **default-outcome** field, enter **default**.
9. Click **Add**, and in the **Outcome** field, enter the name of the email notification page.
10. Use the Expression Builder to enter the following in the **expression** field:

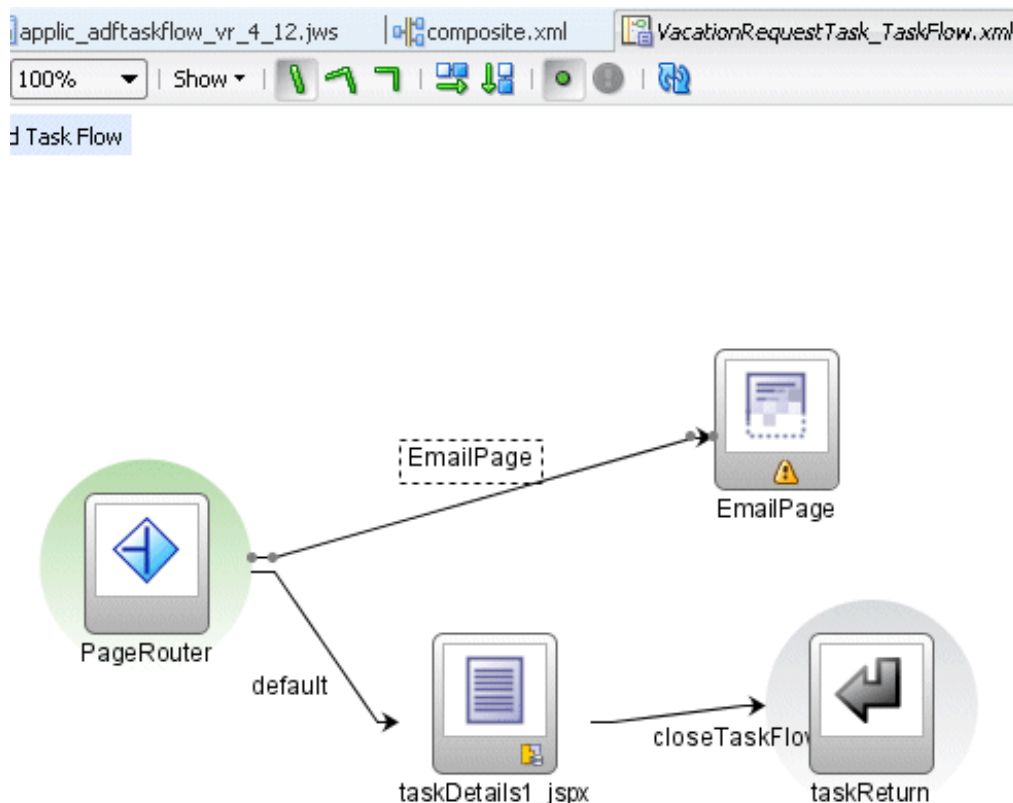
```
{pageFlowScope.bpmClientType=="notificationClient"}
```
11. In the **Component Palette**, click **Control Flow Case**.
12. In the designer, drag from the router page icon to **taskDetails1.jspx**.

The control flow is automatically labeled **default**, as shown in Figure 28–38.

Figure 28–38 Connecting the Control Flow

13. In the **Component Palette**, click **Control Flow Case**.
14. In the designer, drag from the router page icon to the email notification page icon.
15. Click the **control-flow-case - email_page_name - Property Inspector** tab.
16. From the **from-outcome** list, select the name of the email notification page.

Figure 28–39 shows the completed control flow.

Figure 28–39 Completed Control Flow for an Email Notification

To continue creating the email notification page, see Step 1 in [Section 28.7.1.2, "Creating an Email Notification Page."](#)

28.7.1.2 Creating an Email Notification Page

Creating an email notification page is similar to creating a task form, with the addition of defining layout and inline styles. See *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* for design information.

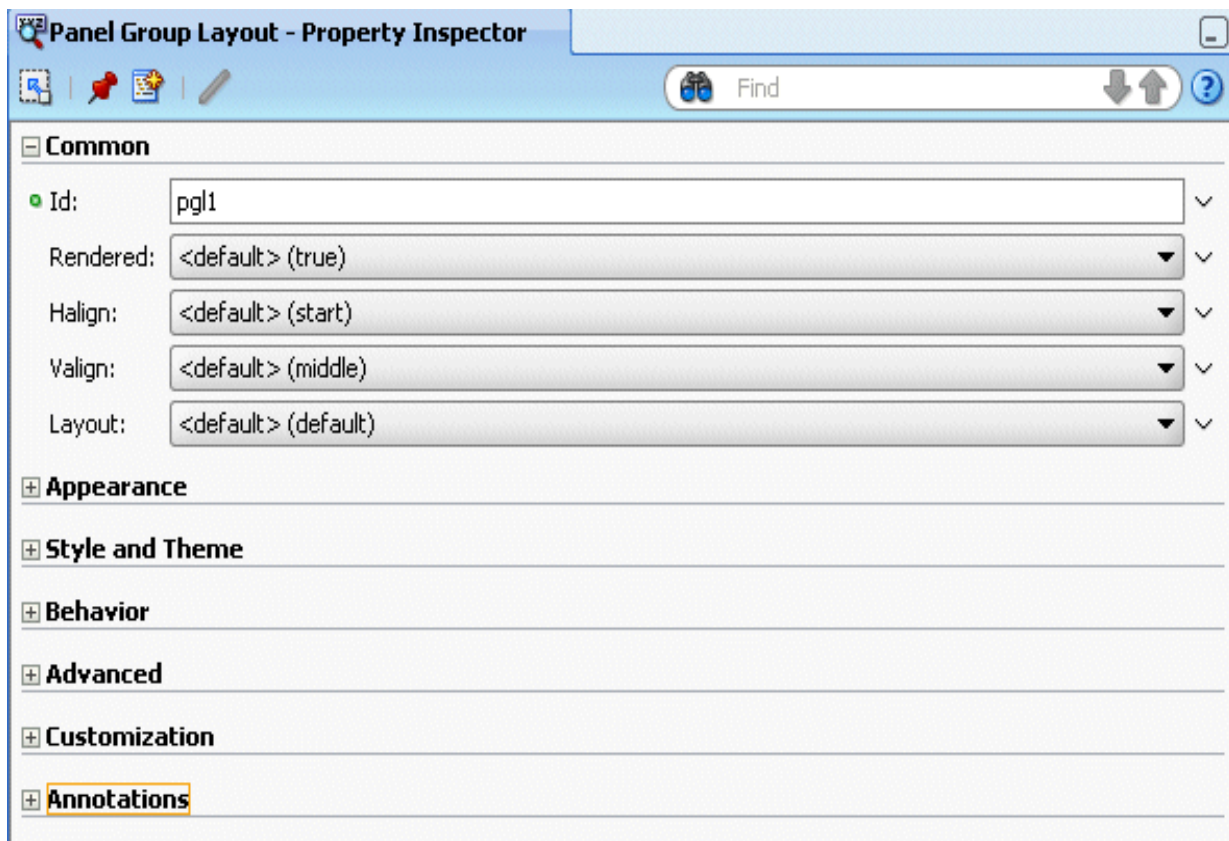
To create an email notification page:

1. In the designer, double-click **EmailPage**.
2. In the Create JSF Page dialog, provide a file name and directory information (or accept the defaults) and click **OK**.

The **EmailPage.jspx** tab opens in the designer.

3. From the **Component Palette**, drag any of the **Common Components** (for an image, for example) or **Layout** components into the designer.
4. For the layout component you selected, provide alignment and other details in the **Property Inspector** tab.

[Figure 28–40](#) shows the layout fields available when **Panel Group Layout** is selected.

Figure 28–40 Specifying a Layout

See *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* for more information about panel group layout.

5. Expand **Appearance**, **Style and Theme**, **Behavior**, **Advanced**, **Customization**, and **Annotations** to specify other details, as shown in [Figure 28–41](#).

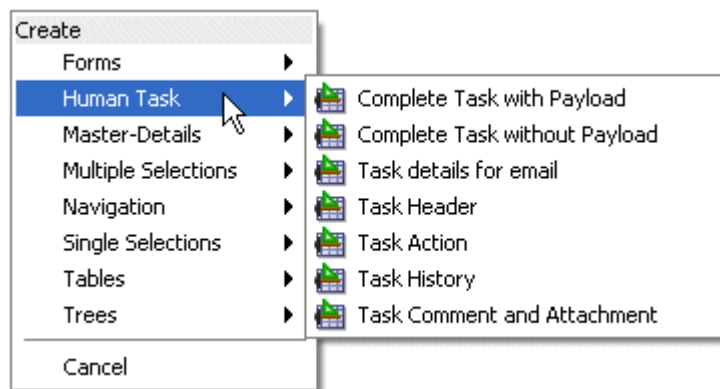
Figure 28–41 Specifying a Layout: More Details

The screenshot shows the 'Panel Group Layout - Property Inspector' window. It has a toolbar with icons for undo, redo, and zoom. The main area is divided into several expandable sections:

- Common**: Contains fields for Id (pgl1), Rendered (<default> (true)), Halign (<default> (start)), Valign (<default> (middle)), and Layout (<default> (default)).
- Appearance**: Contains fields for Halign (<default> (start)), Valign (<default> (middle)), Layout (<default> (default)), ShortDesc, and Rendered (<default> (true)).
- Style and Theme**: Contains fields for Theme, StyleClass, and InlineStyle. Below these are icons for color, font, and size selection, followed by fields for Color, Font, Size (with a percentage dropdown), Italic, Bold, Horizontal Align, Decoration, and Vertical Align (with a percentage dropdown).
- Behavior**: Contains a field for PartialTriggers.
- Advanced**: Contains fields for Binding, ClientComponent (<default> (false)), and AttributeChangeListener.

See "How to Set Component Attributes," in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

6. From the **Data Controls** panel, expand the human task node, then the **getTaskDetails** node, and then the **Return** node.
7. Drag **Task** into the panel group layout area.
8. Select **Human Task**, and then **Task details for email**, as shown in [Figure 28–42](#).

Figure 28–42 Human Task Drop Handlers

This drop handler includes a header with inline style, a payload using ADF, and a comment using inline style. Because the payload is dynamically generated, it does not include an inline style.

In general, you can find the inline styles for the Header section for each component and use the same style for the Content section for the respective components.

9. In the Edit Action Bindings dialog, select the data collection and click **OK**.

The email task form is complete and ready to be deployed.

28.7.2 What Happens When You Create an Email Notification Page

The email notification page is sent as HTML content in the email message body. Images on the page are inlined as attachments. Relative URLs are converted to absolute URLs.

28.7.3 What You May Need to Know About Creating an Email Notification Page

A notification may not display correctly in email if the styles used in the fields of the form are not valid for email. Editing the generated inline CSS to customize the page may be required. See [Section 28.7.1, "How To Create an Email Notification,"](#) for more information.

Security issues can also prevent the form from being rendered correctly. See [Section 28.6, "Securing the Task Flow Application,"](#) for more information.

28.8 Deploying a Composite Application with a Task Flow

The composite application containing the task flow must be deployed before you can use the task form in the Worklist Application. The process for deploying an application with a task flow is basically the same as deploying any SOA composite application, as described in [Section 28.8.1, "How To Deploy a Composite Application with a Task Flow."](#) See [Chapter 40, "Deploying SOA Composite Applications"](#) for more information.

28.8.1 How To Deploy a Composite Application with a Task Flow

An application server connection is required to do the following.

To deploy a composite application with a task flow:

1. Right-click the composite application name, select **Deploy**, and then *application_name* > **to** > *application_server_connection*.

If you do not have a connection, select **New Connection** and use the Application Server Connection wizard.

2. In the **Select Deployment Targets** dialog, select a server instance.
3. Click **OK**.

28.8.2 How To Redeploy the Task Form

If you change the task 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.

28.8.3 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.

28.8.4 How To Deploy a Task Form to a non-SOA Oracle WebLogic Server

This section describes how to deploy a task form to a non-SOA Oracle WebLogic Server.

28.8.4.1 Before Deploying the Task Form: Port Changes

If you are not using the default values for RMI or HTTP ports, open the `wf_client_config.xml` file in Oracle JDeveloper to change values.

When you want to deploy task details on non-SOA servers, you need to configure the `wf_client_config.xml` file. This file should be created and added to the task details project only if the task detail is deployed to a separate managed server that is not the SOA server. The `<serverURL>` and `<rootEndpointURL>` in the file should refer to the SOA server host name and port number.

[Example 28-2](#) shows a sample `wf_client_config.xml` file.

Example 28-2 Sample wf_client_config.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
xmlns="http://xmlns.oracle.com/bpel/services/client">
  <server default="true" name="default">
    <localClient>

<participateInClientTransaction>false</participateInClientTransaction>
    </localClient>
    <remoteClient>
      <serverURL>t3://my_host.us.oracle.com:8001</serverURL>

<initialContextFactory>weblogic.jndi.WLInitialContextFactory</initialContextFactory>
    </remoteClient>
  </server>
</xmlns>
```

```
<soapClient>

<rootEndPointURL>http://my_host.us.oracle.com:8001</rootEndPointURL>
  </soapClient>
</server>
</workflowServicesClientConfiguration>
```

28.8.4.2 Configuring Unique Cookie Context Paths for the Session Tracking Cookies

Before deploying the task form to a non-SOA Oracle Weblogic Server, be sure that the session tracking cookie of your task-form web application is configured with a cookie trigger path unique to your application. This ensures that your task form application has its unique session tracking cookie and cannot be overwritten by the session tracking cookies created for other Oracle BPM applications like Oracle BPM Worklist or Oracle Business Process Management Workspace.

To configure the session cookie trigger path, in JDeveloper, open the `weblogic.xml` file in your web project. Choose the overview tab in your `.xml` file editor, and choose the session. In the cookie trigger path field, enter the application context path of your web application. For example, if the URL of your application is `http://host:port/my-application-context-root` in which `my-application-context-root` is the name of your application context root, then the cookie trigger path is set as follows:

```
/my-application-context-root
```

28.8.4.3 Deploying `oracle.soa.workflow.jar` to a non-SOA Oracle WebLogic Server

The `oracle.soa.workflow.jar` shared library is needed on the non-SOA Oracle WebLogic Server. It is available from

```
ORACLE_JDEV_HOME\jdeveloper\soa\modules\oracle.soa.workflow_11.1.1
```

Use Oracle WebLogic Server Administration Console to deploy the JAR file.

To deploy `oracle.soa.workflow.jar`:

1. Go to Oracle WebLogic Server Administration Console at `http://remote_hostname:remote_portnumber/console`
2. In the **Domain Structure** area, click **Deployments**.
3. Click **Install**, as shown in [Figure 28–43](#).

Figure 28–43 Oracle WebLogic Server Administration Console: List of Deployments

ORACLE WebLogic Server® Administration Console

Home Log Out Preferences Record Help

Home > Summary of Deployments

Summary of Deployments

Control Monitoring

This page displays a list of Java EE applications and stand-alone application modules that have been installed to this domain (redeployed), or deleted from the domain by first selecting the application name and using the controls on this page.

To install a new application or module for deployment to targets in this domain, click the Install button.

Customize this table

Deployments

Install Update Delete Start Stop

☐	Name	State
☐	adf.oracle.domain(1.0,11.1.1.1.0)	Active
☐	adf.oracle.domain.webapp(1.0,11.1.1.1.0)	Active
☐	AqAdapter	Active
☐	b2bui	Installed
☐	DbAdapter	Active
☐	DefaultToDoTaskFlow	Active
☐	DMS Application (11.1.1.1.0)	Active
☐	DocumentReviewTaskFlow	Active
☐	FileAdapter	Active
☐	FtpAdapter	Active

Install Update Delete Start Stop

- In the **Path** field, provide the following path and click **Next**.

```
ORACLE_JDEV_HOME/jdeveloper/soa/modules/oracle.soa.workflow_11.1.1/oracle.soa.workflow.jar
```

- Keep the same name for the deployment and click **Next**, as shown in Figure 28–44.

Figure 28–44 Oracle WebLogic Server Administration Console: Install Applications Assistant

Home > Summary of Deployments

Install Application Assistant

Back Next Finish Cancel

Optional Settings

You can modify these settings or accept the defaults

General

What do you want to name this deployment?

Name:

Security

What security model do you want to use with this application?

☒ **DD Only:** Use only roles and policies that are defined in the deployment descriptors.

☐ **Custom Roles:** Use roles that are defined in the Administration Console; use policies that are defined in the deployment descriptor.

☐ **Custom Roles and Policies:** Use only roles and policies that are defined in the Administration Console.

☐ **Advanced:** Use a custom model that you have configured on the realm's configuration page.

Source accessibility

How should the source files be made accessible?

☒ **Use the defaults defined by the deployment's targets**

Recommended selection.

☐ **Copy this application onto every target for me**

During deployment, the files will be copied automatically to the managed servers to which the application is targeted.

☐ **I will make the deployment accessible from the following location**

- 6. Select the **Deploy as Library** option and click **Finish**.
- 7. Confirm that the oracle.soa.workflow(11.1.1,11.1.1) library is in the **Active** state, as shown in [Figure 28–45](#).

Figure 28–45 Oracle WebLogic Server Administration Console: The oracle.soa.workflow Active State

☐	oracle.soa.workflow(11.1.1,11.1.1)	Active	Library
☐	oracle.soa.worklist(11.1.1,11.1.1)	Active	Library
☐	oracle.wsm.seedpolicies(11.1.1,11.1.1)	Active	Library
☐	OracleAppsAdapter	Installed	Resource Adapter
☐	OracleBamAdapter	Installed	Resource Adapter

See [Section 28.8.4.4, "Defining the Foreign JNDI Provider on a non-SOA Oracle WebLogic Server,"](#) to continue.

28.8.4.4 Defining the Foreign JNDI Provider on a non-SOA Oracle WebLogic Server

Use Oracle WebLogic Server Administration Console to complete this portion of the task.

To define the foreign JNDI provider:

1. In the **Domain Structure** area, expand **Services** and click **Foreign JNDI Providers**.
2. Click **New**.
3. In the **Name** field, enter `ForeignJNDIProvider-SOA`, as shown in [Figure 28-46](#), and click **OK**.

Figure 28-46 *Creating a Foreign JNDI Provider*

Home > Summary of Deployments > Summary of Foreign JNDI Providers

Create a Foreign JNDI Provider

OK Cancel

Foreign JNDI Provider Properties

The following properties will be used to identify your new Foreign JNDI Provider.

* Indicates required fields

What would you like to name your new Foreign JNDI Provider?

* **Name:** ForeignJNDIProvider-SOA

OK Cancel

4. Click the **ForeignJNDIProvider-SOA** link.
5. Do the following and click **Save**.
 - For **Initial Context Factory**, enter `weblogic.jndi.WLInitialContextFactory`.
 - For **Provider URL**, enter `t3://soa_hostname:soa_portnumber/soa-infra`.
In a clustered environment, for **Provider URL**, enter `http://soa_hostname:soa_portnumber/soa-infra`.
 - For **User**, enter `weblogic`.
 - For **Password**, enter `weblogic`.

[Figure 28-47](#) shows the page where you enter this information.

Figure 28–47 Defining the Foreign JNDI Provider

Save

A foreign JNDI provider represents a JNDI tree that can reside outside of a WebLogic Server. This could be a JNDI tree in a different server environment or within an external Java program. By setting up a foreign JNDI provider you can lookup and use an object that exists outside of the WebLogic server environment with the same ease that you would use an object bound in your WebLogic server instance. Use this page to configure a foreign JNDI provider.

Name:	ForeignJNDIProvider-SOA	The user-specified name of this MBean instance. More Info...
Initial Context Factory:		The initial context factory to use to connect. This class name depends on the JNDI provider and the vendor that are being used. The value corresponds to the standard JNDI property, <code>java.naming.factory.initial</code> . More Info...
Provider URL:		The foreign jndi provider url. This value corresponds to the standard JNDI property, <code>java.naming.provider.url</code> . More Info...
User:		The remote server's user name. More Info...
Password:		The remote server's user password. More Info...
Confirm Password:		
Properties:		Any additional properties that must be set for the JNDI provider. These properties will be passed directly to the constructor for the JNDI provider's <code>InitialContext</code> class. More Info...

See [Section 28.8.4.5, "Defining the Foreign JNDI Provider Links on a non-SOA Oracle WebLogic Server,"](#) to continue.

28.8.4.5 Defining the Foreign JNDI Provider Links on a non-SOA Oracle WebLogic Server

Use Oracle WebLogic Server Administration Console to complete this portion of the task.

To define the foreign JNDI provider links:

1. In the **Domain Structure** area, expand **Services** and click **Foreign JNDI Providers**.
2. Click the **ForeignJNDIProvider-SOA** link.
3. Click the **Links** tab.
4. Click **New**.

[Figure 28–48](#) shows the **Links** tab.

Figure 28–48 Defining the Foreign JNDI Provider Links: The Links Tab

Home > Summary of Deployments > Summary of Foreign JNDI Providers > ForeignJNDIProvider-SOA > Summary of Foreign JNDI Providers > **ForeignJNDIProvider-SOA**

Settings for ForeignJNDIProvider-SOA

Configuration Notes

General **Links**

This page lists existing ForeignJNDILinks associated with this ForeignJNDIProvider.

[Customize this table](#)

Foreign JNDI Links

New Clone Delete Showing 0 to 0 of 0 Previous | Next

☐	Name ↕	Local JNDI Name	Remote JNDI Name
There are no items to display			

New Clone Delete Showing 0 to 0 of 0 Previous | Next

5. Do the following and click OK.
 - For **Name**, enter RuntimeConfigService.
 - For **Local JNDI Name**, enter RuntimeConfigService.
 - For **Remote JNDI Name**, enter RuntimeConfigService.

[Figure 28–49](#) shows where you do this.

Figure 28–49 Defining the Foreign JNDI Provider Links: Link Properties

Home > Summary of Deployments > Summary of Foreign JNDI Providers > ForeignJNDIProvider-SOA > Summary of Foreign JNDI Providers > **ForeignJNDIProvider-SOA**

Create a Foreign JNDI Link

OK Cancel

Foreign JNDI Link Properties

The following properties will be used to identify your new Foreign JNDI Link.

* Indicates required fields

What would you like to name your new Foreign JNDI Link?

*Name: RuntimeConfigService

Specify additional properties for this link

Local JNDI Name: RuntimeConfigService

Remote JNDI Name: RuntimeConfigService

OK Cancel

6. Do the following and click OK.
 - For **Name**, **Local JNDI Name**, **Remote JNDI Name**, enter ejb/bpel/services/workflow/TaskServiceBean.

- For **Name, Local JNDI Name, Remote JNDI Name**, enter `ejb/bpel/services/workflow/TaskMetadataServiceBean`.
- For **Name, Local JNDI Name, Remote JNDI Name**, enter `TaskReportServiceBean`.
- For **Name, Local JNDI Name, Remote JNDI Name**, enter `TaskEvidenceServiceBean`.
- For **Name, Local JNDI Name, Remote JNDI Name**, enter `TaskQueryService`.
- For **Name, Local JNDI Name, Remote JNDI Name**, enter `UserMetadataService`.

See [Section 28.8.4.6, "Including a Grant for bpm-services.jar,"](#) to continue.

28.8.4.6 Including a Grant for bpm-services.jar

To include a grant for bpm-services.jar, edit the `system-jazn-data.xml` file and then restart the non-SOA Oracle WebLogic Server.

To include a grant for bpm-services.jar:

1. Locate the `system-jazn-data.xml` file by navigating to the domain directory, `soa-infra`, and then to

`ORACLE_WEBLOGIC_INSTALL/user_projects/domains/your_domain_name/config/fmwconfig`

2. In `system-jazn-data.xml`, add the following grant. (If all or some portion of the grant exists, then add only what is missing.)

```
<grant>
  <grantee>
    <codesource>
      <url>file: ORACLE_JDEV_HOME/jdeveloper/soa/modules/oracle.soa.workflow_
11.1.1/bpm-services.jar</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>VerificationService.createInternalWorkflowContext</name>
    </permission>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission
</class>
      <name>credstoressp.credstore.BPM-CRYPTO.BPM-CRYPTO</name>
      <actions>read,write</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>IdentityAssertion</name>
      <actions>*</actions>
    </permission>
  </permissions>
</grant>
```

3. Restart the non-SOA Oracle WebLogic Server.

See [Section 28.8.4.7, "Deploying the Application,"](#) to continue.

28.8.4.7 Deploying the Application

Deploy the application that contains the task form to a non-SOA Oracle WebLogic Server the same way other applications are deployed. When you set up the application server connection, specify the domain on the non-SOA server (the domain you specified in Step 1 of [Section 28.8.4.6, "Including a Grant for bpm-services.jar"](#)). See [Chapter 40, "Deploying SOA Composite Applications"](#) for information on deploying applications.

28.8.5 What Happens When You Deploy the Task Form

When the task form is deployed, an automatic association is created between the task metadata and the task flow application URL. Use Oracle Enterprise Manager 11g Fusion Middleware Control to update this mapping. Access the task flow component in the **Component Metrics** table for a specific SOA composite application. The Administration tab shows the URI for the task form. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information. If the task flow is configured for HTTPS access, you may need to do additional settings in Enterprise Manager.

See [Chapter 29, "Using Oracle BPM Worklist"](#) for information on how to act on tasks.

Notes:

- For the task form to work correctly, always specify the URL using the complete name for the host on which the task flow is deployed.
 - If you want to access the task form from a different URL that has a different port number than the hostname and port number previously set in Oracle WebLogic Server Administration Console, then you must change the port number for the front-end in Oracle WebLogic Server Administration Console and redeploy the task form so that the task details appear correctly in the worklist.
-
-

28.8.6 What You May Need to Know About Undeploying a Task Flow

When a task flow Web application is deployed, the task flow URL is registered in the database. This URL is displayed in Oracle BPM Worklist when a task is clicked and the task details are displayed. If the task flow Web application is later undeployed or stopped, the task flow URL in the database is *not* removed as part of the undeployment. Consequently, when you click the task in the worklist to see the task details, a 404 Not Found error is displayed rather than the message *Details not available for task*. To avoid the 404 Not Found error, use Oracle Enterprise Manager Fusion Middleware Control to undeploy the task flow application from the application home page.

28.9 Displaying a Task Form in the Worklist

The task 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 28–50](#) shows how the task form for the help desk request example is displayed in the Worklist Application task details page.

Figure 28–50 Worklist Task Details Page

Help desk request for wfaulk

Details

Contents

Location

California

Type

Hardware

Problem Description

Unable to reboot the system

Severity

2

Status

Created

Requester

ID

wfaulk

First Name

William

Last Name

Faulkner

Email

user1@us.oracle.com

Phone

[0]

Resolution

Comment

None

Resolved By

None

History

☒ Task Snapshot
☒ Future Participants
☐ Full task actions

#	Participant	Action	Action Date
1	Stage 1		
1.1	jstein	Assigned	Mar 6, 2009 3:39 PM
1.2	jcooper	RESOLVED, Outcome Updated	Mar 6, 2009 6:32 PM
1.3	jstein	RESOLVED, Outcome Updated	Mar 6, 2009 6:53 PM
1.4	demoadmin	RESOLVED, Alerted	Mar 6, 2009 6:53 PM
1.5	California	RESOLVED, Outcome Updated	Mar 6, 2009 7:04 PM
1.6	jcooper	RESOLVED, Outcome Updated	Mar 6, 2009 7:09 PM
1.7	jstein	RESOLVED, Outcome Updated	Mar 6, 2009 7:11 PM
2	jstein	RESOLVED, Completed	Mar 6, 2009 7:11 PM

Comments

Mar 6, 2009 3:41 PM jstein

I need more info on this

Mar 6, 2009 6:19 PM wfaulk

I have attached a file

Mar 6, 2009 6:32 PM jcooper

I am OK with this

Mar 6, 2009 6:53 PM jstein

Group Vote Comment

Attachments

bugs.xls

28.9.1 How To Display the Task Form in the Worklist

The task form is available in Oracle BPM Worklist after you log in. See [Section 29.2.1, "How To Log In to the Worklist"](#) for instructions.

28.10 Displaying a Task in an Email Notification

[Figure 28–51](#) shows how an email task notification appears in email.

Figure 28–51 Email Task Notification

Subject: Action Required:Help desk request for wfaulk
From: tom@maprao-pc.com
To: dhiraj@maprao-pc.com

Task Help desk request for wfaulk requires your attention.
 Please access the task in the [Worklist Application](#) or take direct action using the links below:

Actions: [RESOLVED](#) [UNRESOLVED](#)



Help desk request for wfaulk

Task Number	200011
State	Assigned
Outcome	Assigned
Priority	3
Created	Mar 16, 2009 6:14 PM
Updated	Mar 16, 2009 6:14 PM
Expiration Date	Mar 18, 2009 6:24 PM
Assignees	jstein

Contents

Location	California
Type	Hardware
Problem Description	Unable to reboot the system
Severity	2
Status	Created

Requester

ID	wfaulk
First Name	William
Last Name	Faulkner
Email	user1@us.oracle.com
Phone	2222222222

Resolution

Comment	None
Resolved By	None

Comments

No rows available

You can click an available action, **RESOLVED** or **UNRESOLVED**, or click the **Worklist Application** link to log in to the worklist. Clicking an action displays an email composer window in which you can add a comment and send the email.

By default, the text in a task notification refers to "Worklist Application", but you can change that text and its associated URL.

28.10.1 Changing the Text for the Worklist Application in Task Notifications

By default, the text in a task notification refers to "Worklist Application", but you can change that text. To do this, you create a custom resource bundle and modify the appropriate strings.

To change the text in a task notification:

1. Open the `WorkflowLabels.properties` resource bundle in the sample `workflow-110-workflowCustomizations`.
2. In the `WorkflowLabels.properties` file, modify the following strings:

```
TASK_NOTIF_MSG.WORKLIST_APPLICATION=Worklist Application
TASK_NOTIF_MSG.WORKSPACE_APPLICATION=Workspace Application
```

For more details on how to modify the resource bundle string, see the `workflow-110-workflowCustomizations` sample.

3. Update the Workflow Custom Classpath URL configuration parameter on your instance.

Note that you do not have to deploy the `WorkflowLabels.properties` file as an application for it to work. Instead, you can do either of the following:

- Host it on the file system, using a URL beginning with `file:///` to point to the appropriate location.
- Host the file in MDS, using a URL beginning with `oramds:///...`

28.10.2 Changing the URL of the Worklist Application in Task Notifications

To change the text in a task notification:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control.
2. In the navigator, expand the **SOA** folder.
3. Right-click **soa-infra**, and select **SOA Administration > Workflow Task Service Properties**.

The Workflow Task Service Properties page appears.

4. Expand **Advanced**.
5. Modify the Worklist Application URL. For example, you can change an existing entry like this:

```
http://[HTTP_HOST]:[HTTP_PORT]/integration/worklistapp/TaskDetails?taskId=PC_
HW_TASK_ID_TAG
```

to something like this:

```
http://[HTTP_HOST]:[HTTP_PORT]/patch/info/page.jspx?taskId=PC_HW_TASK_ID_TAG
```

28.10.3 Showing or Hiding the URL of the Worklist Application in Task Notifications

For information about showing or hiding the URL of the Worklist Application, see [Section 27.3.10.6, "Showing the Oracle BPM Worklist URL in Notifications"](#) on page 27-65.

28.11 Reusing the Task Flow Application with Multiple Human Tasks

You can reuse a single task flow application with multiple human tasks. To use this feature, all human tasks must have identical payload elements.

28.11.1 How To Reuse the Task Flow Application with Multiple Human Tasks

1. Open the `TASKFLOW_PROJ_DIR\adfmsrc\hwtaskflow.xml` file.
2. For each additional human task, add the following element inside the file (at the bottom just before `</hwTaskFlows>`):

```
<hwTaskFlow>
  <WorkflowName>$TASK_NAME</WorkflowName>
  <TaskDefinitionNamespace>$TASK_NAMESPACE</TaskDefinitionNamespace>
  <TaskFlowId>$TASK_FLOW_NAME</TaskFlowId>
  <TaskFlowFileName>$TASK_FLOW_FILENAME</TaskFlowFileName>
</hwTaskFlow>
```

where:

- `$TASK_NAME` is replaced with the name of the human task inside the `.task` file (value of the `<name>` element).
- `$TASK_NAMESPACE` is replaced with the namespace of the human task inside the `.task` file (value of the attribute `targetNamespace` of element `<taskDefinition>`).
- `$TASK_FLOW_NAME` is copied from the existing `<hwTaskFlow>/<TaskFlowId>` element.
- `$TASK_FLOW_FILENAME` is copied from the existing `<hwTaskFlow>/<TaskFlowFileName>` element.

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 includes the following sections:

- [Section 29.1, "Introduction to Oracle BPM Worklist"](#)
- [Section 29.2, "Logging In to Oracle BPM Worklist"](#)
- [Section 29.3, "Customizing the Task List Page"](#)
- [Section 29.4, "Acting on Tasks: The Task Details Page"](#)
- [Section 29.5, "Approving Tasks"](#)
- [Section 29.6, "Setting a Vacation Period"](#)
- [Section 29.7, "Setting Rules"](#)
- [Section 29.8, "Using the Worklist Administration Functions"](#)
- [Section 29.9, "Specifying Notification Settings"](#)
- [Section 29.10, "Using Mapped Attributes \(Flex Fields\)"](#)
- [Section 29.11, "Creating Worklist Reports"](#)
- [Section 29.12, "Accessing Oracle BPM Worklist in Local Languages and Time Zones"](#)
- [Section 29.13, "Creating Reusable Worklist Regions"](#)

For information about how to use the APIs exposed by the workflow service, [Chapter 30, "Building a Custom Worklist Client."](#)

For information about troubleshooting human workflow issues, see section "Human Workflow Troubleshooting" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

29.1 Introduction to Oracle BPM Worklist

Oracle BPM Worklist enables business users to access and act on tasks assigned to them. For example, from a worklist, a loan agent can review loan applications or a manager can approve employee vacation requests. These processes are defined in human tasks.

Oracle BPM Worklist provides different functionality based on the user profile. Standard user profiles include task assignee, supervisor, process owner, and administrator. For example, worklist users can update payloads, attach documents or

comments, and route tasks to other users, in addition to completing tasks by providing conclusions such as approvals or rejections. Supervisors or group administrators can use the worklist to analyze tasks assigned to a group and route them appropriately.

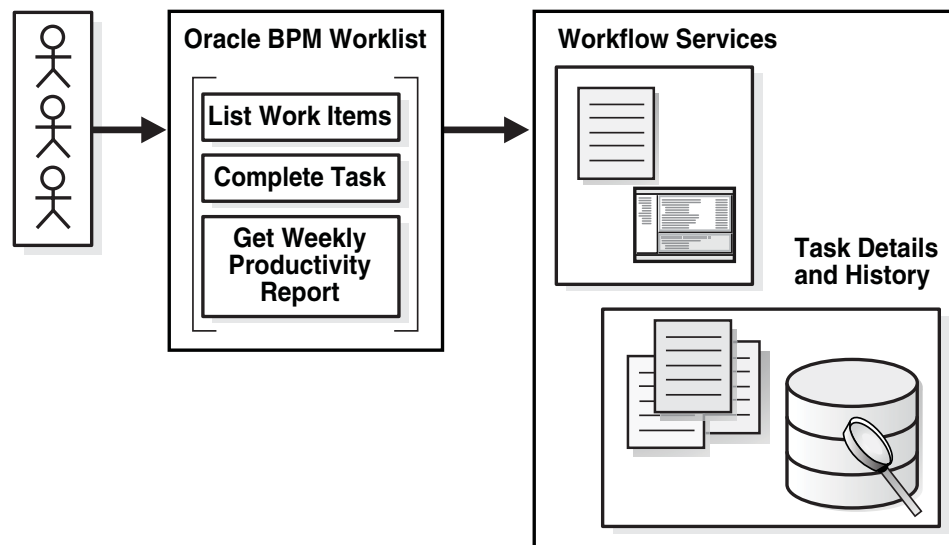
Users can customize their task lists, as required, by adding worklist views, for example, selecting the columns to display, or displaying a subset of the tasks based on filter criteria.

Using Oracle BPM Worklist, task assignees can do the following:

- Perform authorized actions on tasks in the worklist, acquire and check out shared tasks, define personal to-do tasks, and define subtasks.
- Filter tasks in a worklist view based on various criteria.
- Work with standard work queues, such as high priority tasks, tasks due soon, and so on. Work queues allow users to create a custom view to group a subset of tasks in the worklist, for example, high priority tasks, tasks due in 24 hours, expense approval tasks, and more.
- Define custom work queues.
- Gain proxy access to part of another user's worklist.
- Define custom vacation rules and delegation rules.
- Enable group owners to define task dispatching rules for shared tasks.
- Collect a complete workflow history and audit trail.
- Use digital signatures for tasks.

Figure 29–1 shows an illustration of Oracle BPM Worklist.

Figure 29–1 Oracle BPM Worklist—Access Tasks, Forms, Attachments, and Reports



The worklist is rendered in a browser by a task form that you create using ADF task flows in Oracle JDeveloper. See [Chapter 28, "Designing Task Forms for Human Tasks"](#) for more information.

Users can also act on tasks through portals such as Oracle WebCenter. Portals enable users to present information from multiple, unrelated data sources in a single organized view. This view, a portal page, can contain one or more components called portlets that can each collect content from different data sources.

You can build clients for workflow services using the APIs exposed by the workflow service. The APIs enable clients to communicate with the workflow service using local and remote EJBs, SOAP, and HTTP.

29.1.1 What You May Need To Know About Oracle BPM Worklist

Note the following:

- Only one identity provider is supported. Java policy store does not support multiple providers in a sequence. Therefore, fall-through from one directory server to another is not supported for worklists.

29.2 Logging In to Oracle BPM Worklist

Table 29–1 lists the different types of users recognized by Oracle BPM Worklist, based on the privileges assigned to the user.

Table 29–1 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 attributes, and manage rules for any user or group, in addition to his own end-user access.

Note: Multiple authentication providers (for example, SSO and forms) are not supported.

29.2.1 How To Log In to the Worklist

To log in, you must have installed Oracle SOA Suite and the SOA server must be running. See *Oracle Fusion Middleware Installation Guide for Oracle SOA Suite* for more information.

Use a supported web browser:

- Microsoft Internet Explorer 7.x
- Mozilla Firefox 2.x
- Mozilla Firefox 3.x
- Apple Safari

To log in:

1. Go to

`http://hostname:port_number/integration/worklistapp`

- *hostname* is the name of the host computer on which Oracle SOA Suite is installed
 - The *port_number* used at installation
2. Enter the user name and password.

You can use the preseeded user to log in as an administrator. If you have loaded the demo user community in the identity store, then you can use other users such as jstein or jcooper.

The user name and password must exist in the user community provided to JAZN. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for the organizational hierarchy of the demo user community used in examples throughout this chapter.
 3. Click **Login**.

29.2.1.1 Enabling the weblogic User for Logging in to the Worklist

For the weblogic user in Oracle Internet Directory to log in to Oracle BPM Worklist, the Oracle Internet Directory Authenticator must have an Administrators group, and the weblogic user must be a member of that group.

To enable the weblogic user:

1. Create a weblogic user in Oracle Internet Directory using the LDAP browser. The `users.ldif` file is imported to Oracle Internet Directory as follows:

```
dn: cn=weblogic,cn=Users,dc=us,dc=oracle,dc=com
objectclass: inetorgperson
objectclass: organizationalPerson
objectclass: person
objectclass: orcluser
objectclass: orcluserV2
objectclass: top
sn: weblogic
userpassword: welcome1
uid: weblogic
```

2. Create an Administrators group in Oracle Internet Directory and assign the weblogic user to it. The `groups.ldif` file is imported to Oracle Internet Directory as follows:

```
dn: cn=Administrators,cn=Groups,dc=us,dc=oracle,dc=com
objectclass: groupOfUniqueNames
objectclass: orclGroup
objectclass: top
owner: cn=orcladmin,cn=Users,dc=us,dc=oracle,dc=com
uniquemember: cn=weblogic,cn=Users,dc=us,dc=oracle,dc=com
```

29.2.2 What Happens When You Log In to the Worklist

Identity service workflow APIs authenticate and authorize logins using a user name, password, and optionally a realm set, if multiple realms were defined for an organization. See [Section 29.8.2, "How to Specify the Login Page Realm Label,"](#) for information on how administrators can set a preference to change the realm label displayed in the interface, or specify an alternative location for the source of the login page image.

After a user logs in, the **Home** (task list) page displays tasks for the user based on the user's permissions and assigned groups and roles. The **My Tasks** tab and the **Inbox** are displayed by default. The actions allowed from the **Actions** list also depend on the logged-in user's privileges.

Figure 29–2 shows an example of the **Home** page.

Figure 29–2 Oracle BPM Worklist—The Home (Task List) Page

The screenshot displays the Oracle BPM Worklist interface. The top navigation bar includes 'Home', 'Reports', 'Preferences', and 'Help'. The user is logged in as 'js'. The 'My Tasks' tab is active, showing a list of tasks. The left sidebar contains 'Worklist Views' (Inbox, My Work Queues, Standard Views, My Views, Proxy Work Queues, Shared Views) and 'Task Status' (Assigned: 4, Completed: 5, Suspended, Withdrawn, Expired, Errored, Alerted, Info, Requested, Deleted, Stale, Total: 9).

The main task list shows the following tasks:

Title	Number	Priority	Assignees	State	Created	Expires
todo	200005	1	jstein (U)	Assigned	Mar 5, 2009 6:38 PM	
Help desk request for wfaulk	200028	3	jstein (U)	Assigned	Mar 23, 2009 3:24 PM	Mar 24, 2009 3:34 PM
Help desk request for wfaulk	200029	3	jstein (U)	Assigned	Mar 23, 2009 4:22 PM	Mar 24, 2009 4:32 PM
Help desk request for wfaulk	200030	3	jstein (U)	Assigned	Mar 24, 2009 11:09 AM	Mar 25, 2009 11:19 AM

The selected task, 'Help desk request for wfaulk', is shown in detail. The 'Task Actions' are 'RESOLVED' and 'UNRESOLVED'. The 'Details' section shows the task is assigned to 'jstein' with an expiration date of 'Mar 25, 2009 11:19 AM'. The 'Contents' section shows the location as 'California', type as 'Hardware', and problem description as 'Unable to reboot the system'. The 'Requester' section shows the ID as 'wfaulk', first name as 'William', last name as 'Faulkner', email as 'user1@us.oracle.com', and phone as '[0]'. The 'Resolution' section shows the comment as 'None' and resolved by as 'None'. The 'History' section shows a task snapshot with future participants and full task actions. The history table shows the task was assigned to 'jstein' on 'Mar 24, 2009 11:09 AM'.

Table 29–2 describes the components of the **Home** (task list) page.

Table 29–2 Components of the Home (Task List) Page

Component	Description
Tabs	The tabs displayed depend on the role granted to the logged-in user. - 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, and Evidence Search tabs. - Users who are managers and administrators see all the tabs— My Tasks, Initiated Tasks, My Staff Tasks, Administration Tasks, Administration, and Evidence Search. - Users with the workflow.admin.evidenceStore permission also see the Evidence Search tab. See the following for more information: Section 29.4.4, "How To Act on Tasks That Require a Digital Signature," for information about evidence search Section 29.8.1, "How To Manage Other Users' or Groups' Rules (as an Administrator)"
Worklist Views	Inbox, My Work Queues, Proxy Work Queues —See Section 29.3.2, "How To Create and Customize Worklist Views," for more information.
Task Status	A bar chart shows the status of tasks in the current view. See Section 29.3.3, "How To Customize the Task Status Chart," for more information.
Display Filters	Specify search criteria from the View, Assignee or Status fields. The category filters that are available depend on which tab is selected. - The View filters are Inbox, Due Soon, High Priority, and New Tasks. - From the My Tasks tab, the Assignee filters are My, Group, My & Group, Previous (tasks worked on previously), and Reviewer. From the Initiated Tasks tab, the only assignee filter is Creator. From the My Staff Tasks tab, the only assignee filter is Reportees. From the Administration Tasks tab, the only assignee filter is Admin. - The Status filters include Any, Assigned, Completed, Suspended, Withdrawn, Expired, Errored, Alerted, Information Requested. Use Search to enter a keyword, or use Advanced Search. See Section 29.3.1, "How To Filter Tasks," for more information.

Table 29–2 (Cont.) Components of the Home (Task List) Page

Component	Description
Actions List	Select a group action (Claim) or a custom action (for example, Approve or Reject) that was defined for the human task. Claim appears for tasks assigned to a group or multiple users; one user must claim the task before it can be worked. Other possible actions for a task, such as system actions, are displayed on the task details page for a specific task. You can also create ToDo tasks and subtasks here. Note: ■ If a task is aggregated, you only see actions such as Approve and Reject, even if the aggregated task includes FYI tasks. No acknowledge action is explicitly provided. Approve or Reject can be interpreted as an acknowledge action. ■ The Claim button remains enabled even when Auto Claim has been previously enabled. This button enables a user to claim and continue working on the task rather than to simply approve it.
Default Columns	Title—The title specified when the human task was created. Tasks associated with a purged or archived process instance do not appear. Number—The task number generated when the BPEL process was created. Priority—The priority specified when the human task was created. The highest priority is 1; the lowest is 5. Assignees—The user or group or application roles. State—Select from Assigned, Completed, Errored, Expired, Information Requested, Stale, Suspended, or Withdrawn. Created—Date and time the human task was created Expires—Date and time the tasks expires, specified when the human task was created
Task Details	The lower section of the worklist displays the inline view of the task details page. Buttons indicate available actions. See Section 29.4, "Acting on Tasks: The Task Details Page," for more information.

[Figure 29–2](#) also shows the **Administration**, **Reports**, and **Preferences** links (upper-right corner). [Table 29–3](#) summarizes the **Home**, **Administration**, **Reports**, and **Preferences** pages.

Table 29–3 Worklist Main Pages Summary

Page	Description
Home	As described in Table 29–2 , the logged-in user's list of tasks, details for a selected task, and all the functions needed to start acting on a task are provided.
Administration	The following administrative functions are available: ■ Setting application preferences ■ Mapping attributes ■ Searching the evidence store ■ Configuring tasks
Reports	The following reports are available: Unattended Tasks Report, Tasks Priority Report, Tasks Cycle Time Report, Tasks Productivity Report, and Tasks Time Distribution Report. See Section 29.11.1, "How To Create Reports," for more information.
Preferences	Preference settings include: ■ Setting rules for users or groups, including vacation rules, and setting vacation periods ■ Uploading certificates ■ Specifying user notification channels and message filters

29.2.3 What Happens When You Change a User's Privileges While They are Logged in to Oracle BPM Worklist

If you change a user's privileges in Oracle Enterprise Manager Fusion Middleware Control while the user is logged in to Oracle BPM Worklist, the changes take effect only after a subsequent login by the user. This is true for situations in which there are two active worklist sessions, one in which the user is logged in before the privileges are changed, and one in which the same user logs in after the privileges are changed. In the first case, the changes to the user's privileges do not take effect while the user is logged in. In the second case, when the user logs in to the second instance of the Worklist Application, the changes to the user's privileges do take effect.

29.3 Customizing the Task List Page

You can customize your task list in several ways, including adding worklist views, selecting which columns to display, and displaying a subset of the tasks based on filter criteria. Resize the task list display area to increase the number of tasks fetched.

Note: When you deploy SOA composite applications with human tasks to partitions, the tasks created for these composites cannot be filtered using the partition as a parameter inside Oracle BPM Worklist. For example, you can select a task type corresponding to a particular partition (the same task type, but in different partitions), but filtering does not work with the advanced search, custom views, custom rules, and mapped attribute features. For example, assume VacationRequestApp is deployed to partition 1 and partition 2. When the advanced search is used to select tasks corresponding to composites deployed in partition 1, the result does not return the tasks.

29.3.1 How To Filter Tasks

Figure 29–3 shows the filter fields.

Figure 29–3 Filters—Assignee, Status, Search, and Advanced Search

Actions ▼		Assignee	My & Group ▼	Status	Assigned ▼	Search		Advanced
Title		Number	Priority	Assignees	State	Created		
 Vacation Request for jcooper		200203	3	jstein (U)	Assigned	Mar 16, 2009 2:16 PM		

Filters are used to display a subset of tasks, based on the following filter criteria:

- **Assignee**

From the **My Tasks** tab, select from the following:

- **My**—Retrieves tasks directly assigned to the logged-in user
- **Group**—Retrieves the following:
 - * Tasks that are assigned to groups that the logged-in user belongs to
 - * Tasks that are assigned to an application role that the logged-in user is assigned

* Tasks that are assigned to multiple users, one of which is the logged-in user

- **My & Group**—Retrieves all tasks assigned to the user, whether through direct assignment, or by way of a group, application role, or list of users
- **Previous**—Retrieves tasks that the logged-in user has updated
- **Reviewer**—Retrieves task for which the logged-in user is a reviewer

From the **Initiated Tasks** tab, select **Creator**.

From the **My Staff Tasks** tab, select **Reportees**.

From the **Administration Tasks** tab, select **Admin**.

- **Status**—Select from the following: **Any**, **Assigned**, **Completed**, **Suspended** (can be resumed later), **Withdrawn**, **Expired**, **Errored** (while processing), **Alerted**, or **Information Requested**.
- **Search**—Enter a keyword to search task titles, comments, identification keys, and the flex string fields of tasks that qualify for the specified filter criterion.
- **Advanced**—Provides additional search filters.

Note: If a task is assigned separately to multiple reportees, when a manager looks at the My Staff Tasks list, the manager sees as many copies of that task as the number of reportees that the task is assigned to.

To filter tasks based on assignee or status:

1. Select options from the **Assignee** and **Status** lists.

The task list is automatically updated based on the filter selections.

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. Press **Enter** or click **Refresh**.

To filter tasks based on an advanced search:

Mapped attribute labels can be used in an advanced search if you select task types for which mapped attribute mappings have been defined.

See [Section 29.10.1, "How To Map Attributes,"](#) for more information.

1. Click **Advanced**.
2. (Optional) Check **Save As View**, provide a view name, and use the **Display** tab to provide other information, as shown in [Figure 29-4](#) and [Figure 29-5](#).

Figure 29–4 Worklist Advanced Search—Definition Tab

Advanced Search

Definition

Display

☒ Save Search as View

* Name

Assignee

Add Condition

Match ☐ All ☒ Any

Task Type

Share View ☐ Definition only ☒ Data

Assignees

Figure 29–5 Worklist Advanced Search—Display Tab

Advanced Search

Definition

Display

Column 'Title' would always be present and would be first in the task table.

Select View Columns

* Available

☐ Start Date

☐ Task Definition Name

☐ Owner Role

☐ Updated Date

☐ Composite Version

☐ Creator

☐ From User

☐ Percentage Complete

☐ Owner Group

☐ End Date

Move

Move All

Remove

Remove All

Selected

☐ Title

☐ Number

☐ Priority

☐ Assignees

☐ State

☐ Created

☐ Expires

Move

Move All

Remove

Remove All

Sort By Column

Sort Order

Table 29–4 describes the advanced search view columns available in the **Display** tab.

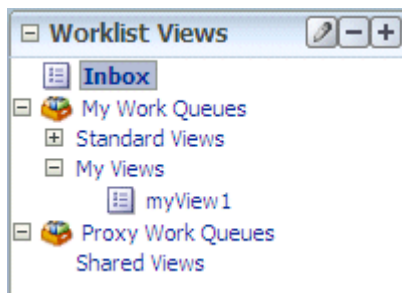
Table 29–4 Advanced Search—View Columns

Column	Description
Start Date	The start date of the task (used with ToDo tasks).
Task Definition Name	The name of the task component that defines the task instance.
Owner Role	The application role (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is an application role, this field is set.
Updated Date	The date the task instance was last updated.
Composite Version	The version of the composite that contains the task component that defines the task instance.
Creator	The name of the creator of the task.
From User	The from user for the task.
Percentage Complete	The percentage of the task completed (used with ToDo tasks).
Owner Group	The group (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is a group, this field is set.
End Date	The end date of the task (used with ToDo tasks).
Composite	The name of the composite that contains the task component that defines the task instance.
Due Date	The due date of the task (used with ToDo tasks).
Composite Distinguished Name	The unique name for the particular deployment of the composite that contains the task component that defines the task instance.
Task Display URL	The URL to display the details for the task.
Updated By	The user who last updated the task.
Outcome	The outcome of the task, for example Approved or Rejected. This is only set on completed task instances.
Task Namespace	A namespace that uniquely defines all versions of the task component that defines this task instance. Different versions of the same task component can have the same namespace, but no two task components can have the same namespace.
Approvers	The approvers of the task.
Application Context	The application to which any application roles associated with the tasks (such as assignees, owners, and so on) belong.
Owner User	The user (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is a user, this field is set.
Identifier	The (optional) custom unique identifier for the task. This is an additional unique identifier to the standard task number.
Category	The category of the task.
Acquired By	The name of the user who claimed the task in the case when the task is assigned to a group, application role, or to multiple users, and then claimed by the user.
Component	The name of the task component that defines the task instance.
Original Assignee User	The name of the user who delegated the task in the case when the user delegates a task to another user.
Assigned	The date that this task was assigned.

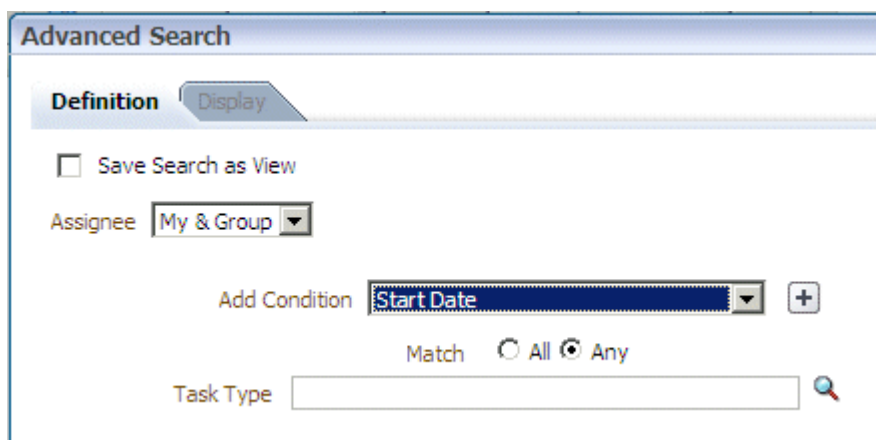
Table 29–4 (Cont.) Advanced Search—View Columns

Column	Description
Domain	The domain to which the composite that contains the task component that defines the task instance belongs.
Title	The title of the task.
Number	An integer that uniquely identifies the task instance.
Priority	An integer that defines the priority of the task. A lower number indicates a higher priority—typically numbers 1 to 5 are used.
Assignees	The current task assignees (users, groups or application roles).
State	The state of the task instance.
Created	The date that the task instance was created.
Expires	The date on which the task instance expires.

The saved view appears in the **Inbox** under **My Views**, as shown in [Figure 29–6](#).

Figure 29–6 Saving a View

3. Select an assignee, as shown in [Figure 29–7](#).

Figure 29–7 Worklist Advanced Search

4. Add conditions (filters), as shown in [Figure 29–8](#).

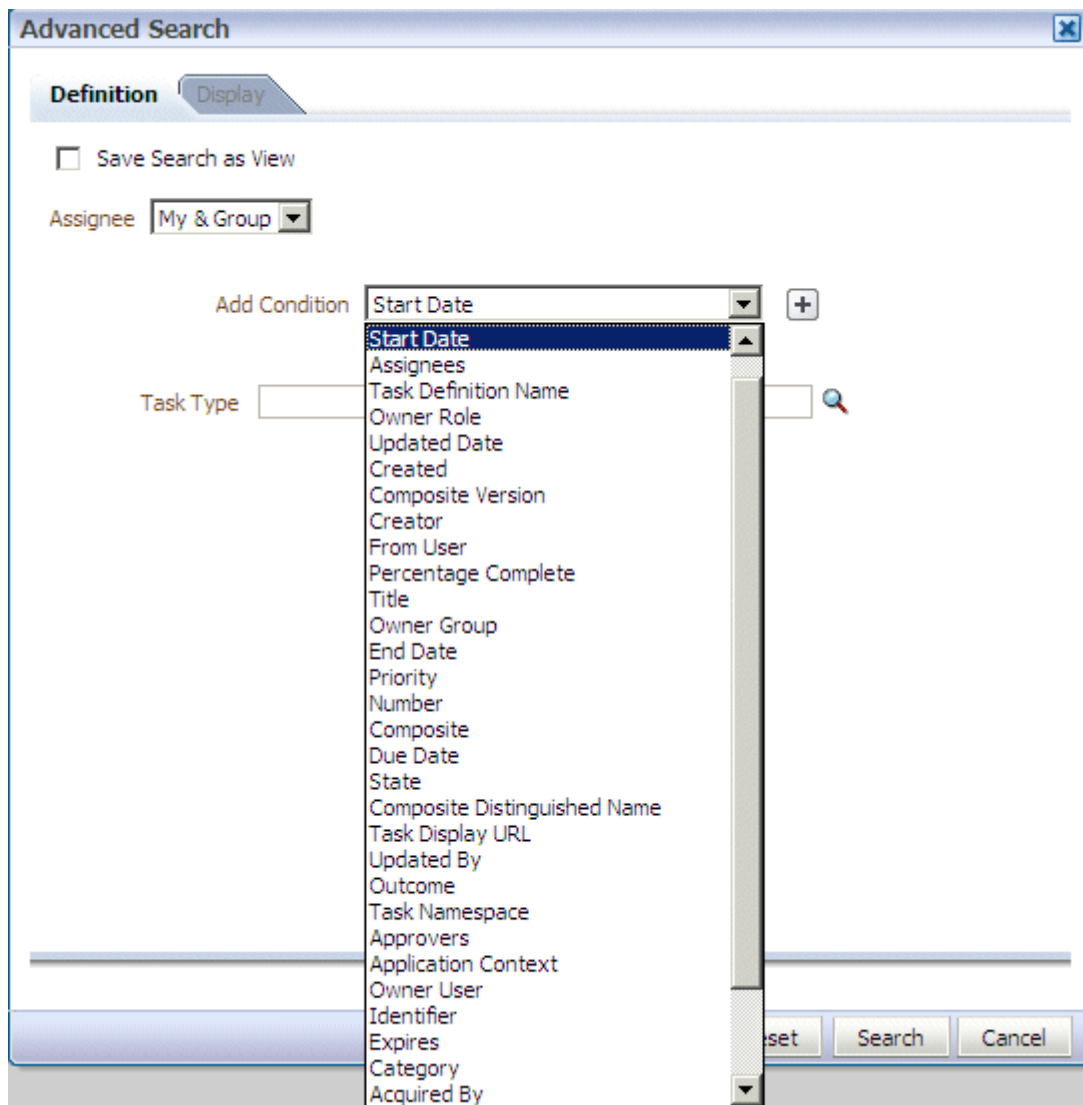
Figure 29–8 Adding Filters for an Advanced Search on Tasks

Table 29–5 describes the available conditions.

Table 29–5 Advanced Search—Conditions

Condition	Description
Start Date	The start date of the task (used with ToDo tasks).
Assignees	The current task assignees (users, groups or application roles).
Task Definition Name	The name of the task component that defines the task instance.
Owner Role	The application role (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is an application role, this field is set.
Updated Date	The date that the task instance was last updated.
Created	The date that the task instance was created.
Composite Version	The version of the composite that contains the task component that defines the task instance.
Creator	The name of the creator of the task.

Table 29–5 (Cont.) Advanced Search—Conditions

Condition	Description
From User	The from user for the task.
Percentage Complete	The percentage of the task completed (used with ToDo tasks).
Title	The title of the task.
Owner Group	The group (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is a group, this field is set.
End Date	The end date of the task (used with ToDo tasks).
Priority	An integer that defines the priority of the task. A lower number indicates a higher priority—typically numbers 1 to 5 are used.
Number	An integer that uniquely identifies the task instance.
Composite	The name of the composite that contains the task component that defines the task instance.
Due Date	The due date of the task (used with ToDo tasks).
State	The state of the task instance.
Composite Distinguished Name	The unique name for the particular deployment of the composite that contains the task component that defines the task instance.
Task Display URL	The URL to display the details for the task.
Updated By	The user who last updated the task.
Outcome	The outcome of the task, for example Approved or Rejected. This is only set on completed task instances.
Task Namespace	The namespace of the task.
Approvers	The approvers of the task.
Application Context	The application to which any application roles associated with the tasks (such as assignees, owners, and so on) belong.
Owner User	The user (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is a user, this field is set.
Identifier	The (optional) custom unique identifier for the task. This is an additional unique identifier to the standard task number.
Expires	The date on which the task instance expires.
Category	The category of the task.
Acquired By	The name of the user who claimed the task in the case when the task is assigned to a group, application role, or to multiple users, and then claimed by the user.
Component	The name of the task component that defines the task instance.
Original Assignee User	The name of the user who delegated the task in the case when the user delegates a task to another user.
Assigned	The date that this task was assigned.
Domain	The domain to which the composite that contains the task component that defines the task instance belongs.

5. Add parameter values, shown in [Figure 29–9](#).

Figure 29–9 Advanced Search

Advanced Search

Definition | Display

☐ Save Search as View

Assignee: My & Group

Add Condition: Start Date +

Match: ☐ All ☒ Any

Task Type:

Start Date: on

Reset Search Cancel

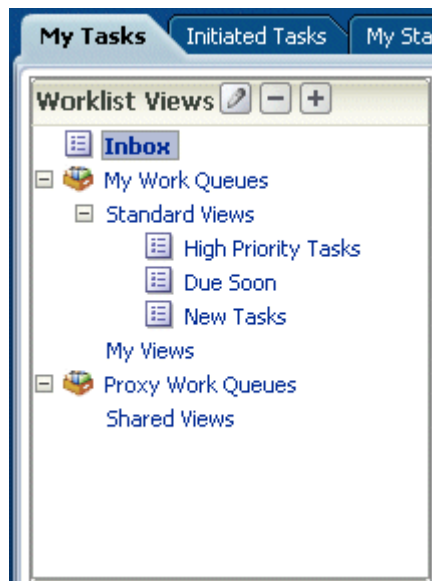
6. Select **Any** or **All** for matching multiple filters.
7. (Optional) Search on a task type.
8. Click **Search**.

The task list page with the tasks filtered according to your criteria appears.

29.3.2 How To Create and Customize Worklist Views

The **Worklist Views** area, shown in [Figure 29–10](#), displays the following:

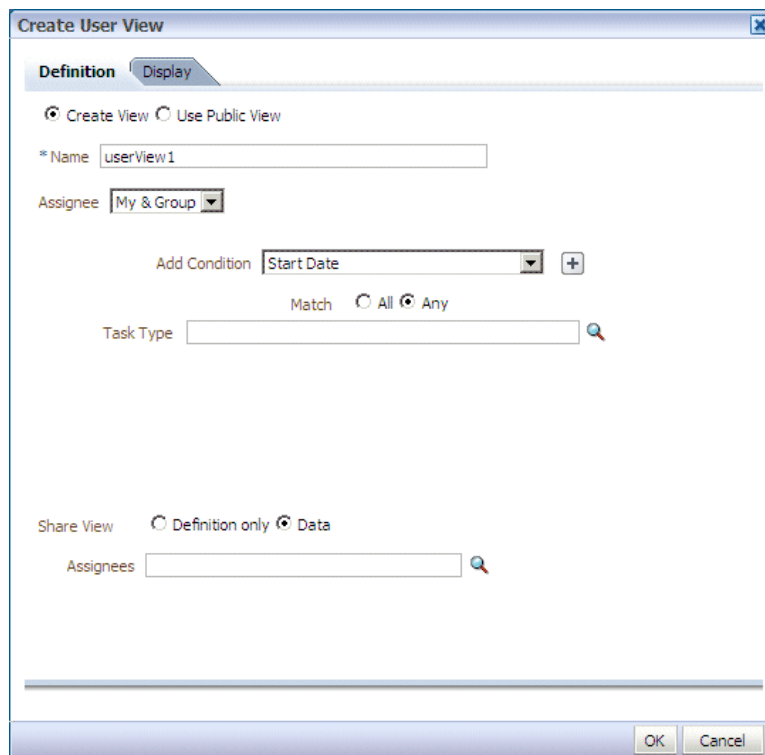
- **Inbox**—Shows all tasks that result from any filters you may have used. The default shows all tasks.
- **My Work Queues**—Shows standard views and views that you defined.
- **Proxy Work Queues**—Shows shared views.

Figure 29–10 Worklist Views

Use **Worklist Views** to create, share, and customize views.

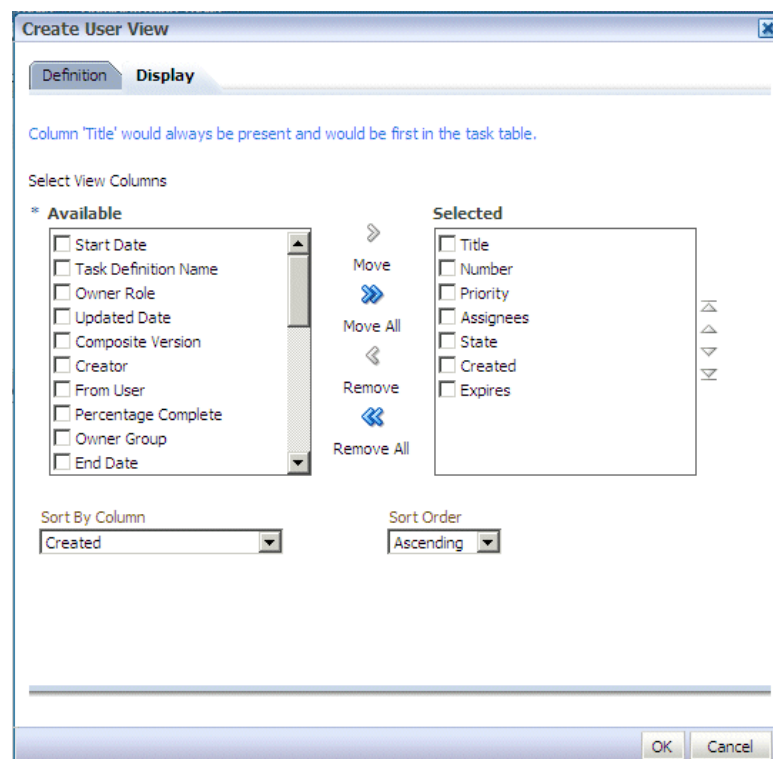
To create a worklist view:

1. In the **Worklist Views** section, click **Add**.
2. Use the **Definition** tab of the Create User View dialog, shown in [Figure 29–11](#).

Figure 29–11 Creating a Worklist View

- **Create View or Use Public View**—Create your own view or browse for a public view to copy.
 - **Name**—Specify a name for your view.
 - **Add to Standard Views**—This option applies to Administrators only. Administrators select this option to create the view as a standard view, which then appears in the **Standard Views** list for *all* worklist users.
 - **Assignee**—Select **My**, **Group**, **My&Group**, **Previous**, or **Reviewer**.
 - **Add Condition**—Select a filter from the list and click **Add**. For example, if you select **startDate**, and click **Add**, then a calendar and a list including **on**, **equals**, **not equals**, **greater than**, **less than**, and so on appears.
 - **Task Type**—Browse for a task type or leave the field blank for all types. Mapped attribute labels can be selected in the query and display columns dialogs if the selected task types have mapped attribute mappings defined.
 - **Match**—Select **All** or **Any** to match the conditions you added.
 - **Share View**—You can grant access to another user to either the definition of this view, in which case the view conditions are applied to the grantee's data, or to the data itself, in which case the grantee can see the grantor's worklist view, including the data. Sharing a view with another user is similar to delegating all tasks that correspond to that view to the other user; that is, the other user can act on your behalf. Shared views are displayed under **Proxy Work Queues**.
 - **Assignees**—Specify the users (grantees) who can share your view.
3. Use the **Display** tab of the Create User View dialog, shown in [Figure 29–12](#), to customize the fields that appear in the view.

Figure 29–12 *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 mapped attributes that have been mapped for the specific task type. The default columns are the same as the columns in your inbox.
 - **Sort by Column**—Select a column to sort on.
 - **Sort Order**—Select ascending or descending order.
4. Click **OK**.

To customize a worklist view:

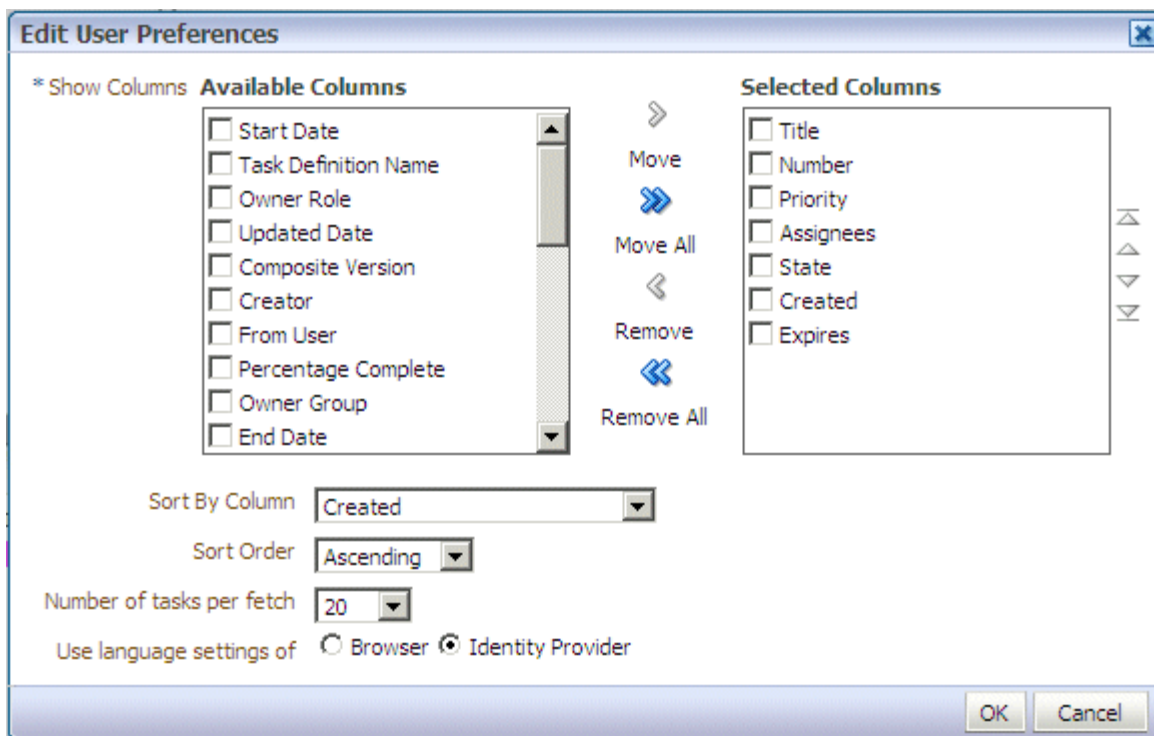
1. In the **Worklist Views** section, click the view name.
2. Click the **Edit** icon.
3. Use the **Definition** and **Display** tabs of the Edit User View dialog to customize the view, as shown in [Figure 29–13](#) and [Figure 29–14](#), and click **OK**.

Figure 29–13 Customizing a Worklist View

The screenshot shows the 'Edit User View : myView1' dialog box with the 'Definition' tab selected. The dialog contains the following elements:

- Name:** A text field containing 'myView1'.
- Assignee:** A dropdown menu showing 'My'.
- Add Condition:** A section with a dropdown menu showing 'Start Date' and a '+' button.
- Match:** Radio buttons for 'All' (selected) and 'Any'.
- Task Type:** A text field with a search icon.
- Assignees:** A dropdown menu showing 'equals' and a text field containing 'j', with a '-' button.
- Share View:** Radio buttons for 'Definition only' and 'Data' (selected).
- Assignees:** A text field with a search icon.
- Buttons:** 'OK' and 'Cancel' buttons at the bottom right.

Figure 29–14 Customizing Fields in a Worklist View



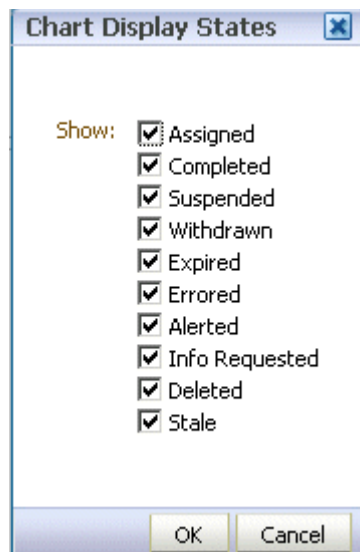
When you select and move items from the **Available Columns** list to the **Selected Columns** list (or vice-versa), the items remain checked. Therefore, if you select items to move back, the previously selected items are also moved. Be sure to uncheck items after moving them between the lists if you intend to move additional columns.

29.3.3 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.

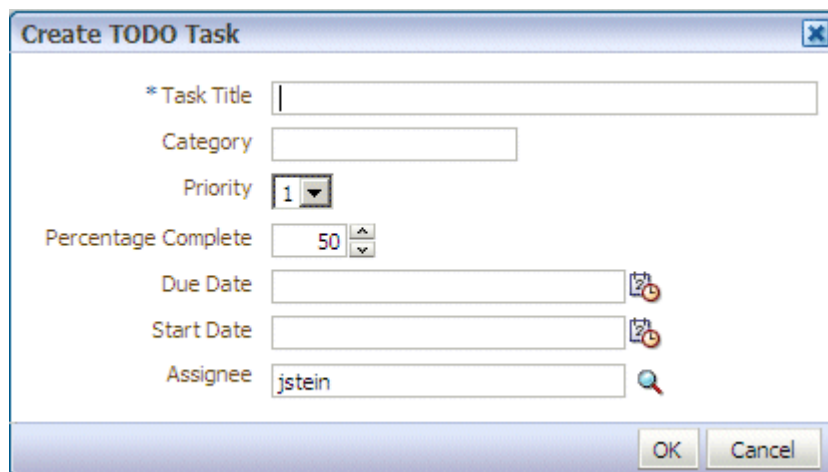
To customize the task status chart:

1. Click the **Edit** icon.
2. Add or remove status states for display, as shown in [Figure 29–15](#), and click **OK**.

Figure 29–15 Customizing the Task Status Chart

29.3.4 How To Create a ToDo Task

Use the Create ToDo Task dialog, shown in [Figure 29–16](#), to create a top-level ToDo task for yourself or others. This task is not associated with a business task.

Figure 29–16 The Create ToDo Task Dialog

ToDo tasks appear in the assignee's **Inbox**.

You can create ToDo tasks that are children of other ToDo tasks or business tasks. A ToDo task can have only one level of child ToDo tasks. When all child ToDo tasks are 100% complete, the parent ToDo task is also marked as completed. If the parent ToDo task is completed, then child ToDo tasks are at 100% within the workflow system. If the parent is a business task, the child ToDo is not marked as completed. You must set the outcome and complete it. If you explicitly set a ToDo task to 100%, there is no aggregation on the parent task.

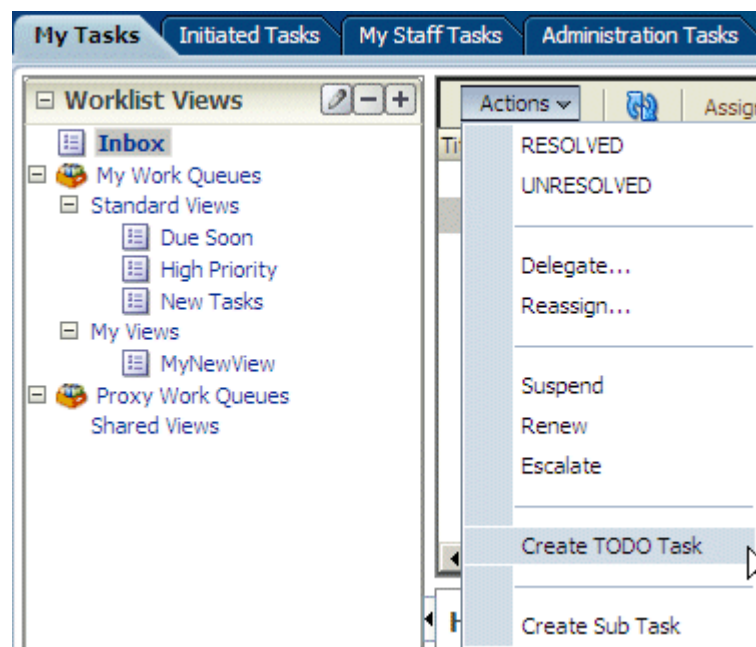
ToDo tasks can be reassigned, escalated, and so on, and deleted (logical delete) and purged (physical delete). Reassignment, escalation, and so on of the parent task does not affect the assignment of any child ToDo tasks. The completion percentage of a ToDo task can be reset to less than 100% after it is completed.

Assignment rules (such as vacation rules) are not applied to ToDo tasks. You cannot specify business rules for ToDo tasks.

To create a ToDo task:

1. From the **Actions** list, select **Create TODO Task**, as shown in [Figure 29–17](#).

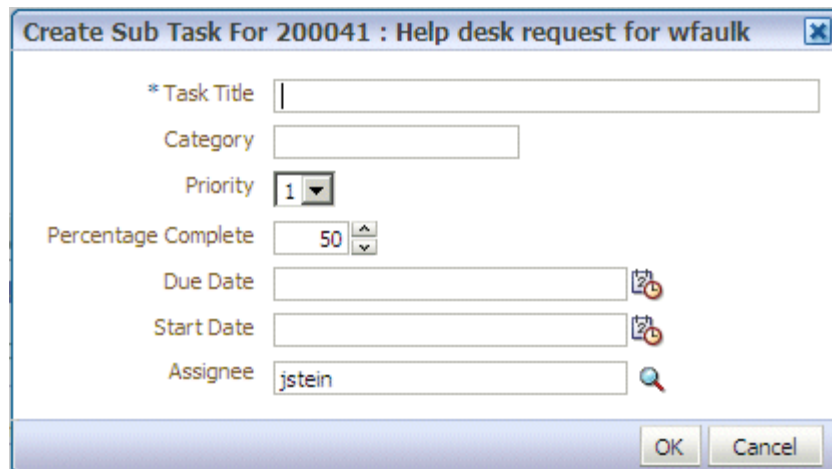
Figure 29–17 *Creating a ToDo Task*



2. Provide details in the Create ToDo Task dialog, shown in [Figure 29–17](#), and click **OK**.
 - **Task Title:** Enter anything that is meaningful to you.
 - **Category:** Enter anything that is meaningful to you.
 - **Priority:** Select from **1** (highest) to **5** (lowest)
 - **Percentage Complete:** This attribute indicates how much of the task is completed. 100% sets the attribute as completed.
 - **Due Date:** The due date does not trigger an expiration. You can also see overdue tasks. The start date need not be the current date.
 - **StartDate:** The task start date.
 - **Assignee:** You can assign yourself or someone else.

29.3.5 How To Create a Subtask

Use the Create Sub Task dialog, shown in [Figure 29–18](#), to create a subtask, which is a ToDo task for a business task. You must select a business task before selecting the **Create Sub Task** option (shown in [Figure 29–17](#)).

Figure 29–18 Creating a Subtask

The screenshot shows a dialog box titled "Create Sub Task For 200041 : Help desk request for wfaulk". It contains the following fields and controls:

- *Task Title**: A text input field.
- Category**: A text input field.
- Priority**: A dropdown menu with the value "1" selected.
- Percentage Complete**: A spinner control with the value "50".
- Due Date**: A text input field with a calendar icon.
- Start Date**: A text input field with a calendar icon.
- Assignee**: A text input field with the value "jstein" and a search icon.
- Buttons**: "OK" and "Cancel" buttons at the bottom right.

Subtasks can break down a business task into measurable subtasks, and can be created for ToDo tasks also. Multiple levels of subtasks are not supported (that is, you cannot have subtasks inside of subtasks). If you create multiple levels of subtasks, and attempt to act on the main task (for example, to approve or reject), you receive an error.

29.4 Acting on Tasks: The Task Details Page

Task details can be viewed inline (see the lower section in [Figure 29–2, "Oracle BPM Worklist—The Home \(Task List\) Page"](#)) or in a pop-up browser window. (Double-click the task.)

[Figure 29–19](#) shows the task details page.

Figure 29–19 Task Details Page

Help desk request for wfaulk

Task Actions ▾

RESOLVED

UNRESOLVED

Details

Assignees

Creator

Created Apr 6, 2009 4:43 PM

Updated Apr 6, 2009 4:43 PM

Expiration date

Apr 7, 2009 4:53 PM

Acquired By

Task Number 200040

Priority 3

State

Assigned

Contents

Location

California

Type

Hardware

Problem Description

Unable to reboot the system

Severity

2

Status

Created

Requester

ID

wfaulk

First Name

William

Last Name

Faulkner

Email

user1@us.oracle.com

Phone

[0]

Resolution

Comment

None

Resolved By

None

History

Task Snapshot

☒ Future Participants

☐ Full task actions

#	Participant	Action	Action Date
1	Stage1		
1.1	jstein	Assigned	Apr 6, 2009 4:43 PM

Comments

Attachments

+

×

Any kind of change to the task details page, such as changing a priority or adding a comment or attachment, requires you to save the change before you go on to make any other changes.

The task details page has the following components:

- **Task Actions**—Lists the system actions that are possible for the task, such as **Request Information**, **Reassign**, **Renew**, **Suspend**, **Escalate**, and **Save**.
- **Action buttons**—Displays buttons for custom actions that are defined in the human task, such as setting task outcomes (for example, **Resolved** and **Unresolved** for a help desk request or **Approve** and **Reject** for a loan request). For the task initiator, manager, or administrator, **Withdraw** may also appear.

- **Details**—Displays task attributes, including the assignee, task creator, task number, state, priority, who acquired the task, and other mapped attributes. It also displays dates related to task creation, last update, and expiration date.
- **Contents**—Displays the payload. The fields displayed are specific to how the human task was created.
- **Requester**—Displays details (full name, contact information, and so on) about the task requester.
- **Resolution**—Displays any comments or resolution status.
- **History**—Displays the approval sequence and the update history for the task. See [Section 29.4.2, "Task History,"](#) for more information.

[Table 29–6](#) tells what the icons used in the Task Details History section signify.

Table 29–6 Icons for Task Action History

Icon	Description
	Indicates an approver in an ad hoc routing scenario.
	Indicates that the task has been approved.
	Indicates that the participant is an FYI participant—that is, this participant just receives a notification task and the business process does not wait for the participant's response. Participant cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.
	Indicates that a set of people must work in parallel. This pattern is commonly used for voting.
	Indicates that the participant belongs to a management chain.
	Indicates the simple case in which a participant maps to a user, group, or role.
	Indicates that the task is untouched.

- **Comments**—Displays comments entered by various users who have participated in the workflow. A newly added comment and the commenter's user name are appended to the existing comments. A trail of comments is maintained throughout the life cycle of the task. To add or delete a comment, you must have permission to update the task.
- **Attachments**—Displays documents or reference URLs that are associated with a task. These are typically associated with the workflow as defined in the human task or attached and modified by any of the participants using the worklist. To add or delete an attachment, you must have permission to update the task. When adding file attachments, you can use an absolute path name or browse for a file.

Comments and attachments are shared between tasks and subtasks. Therefore, when you create a ToDo task and add comments and attachments, subtasks of this ToDo task include the same comments and attachments.

A user can view a task when associated with the task as the current assignee (directly or by group membership), the current assignee's manager, the creator, the owner, or a previous actor.

A user's profile determines his group memberships and roles. The roles determine a user's privileges. Apart from the privileges, the exact set of actions a user can perform is also determined by the state of the task, the custom actions, and restricted actions defined for the task flow at design time.

The following algorithm is used to determine the actions a user can perform on a task:

1. Get the list of actions a user can perform based on the privileges granted to him.
2. Get the list of actions that can be performed in the current state of the task.
3. Create a combined list of actions that appear on the preceding lists.
4. Remove any action on the combined list that is specified as a restricted action on the task.

The resulting list of actions is displayed in the task list page and the task details page for the user. When a user requests a specific action, such as claim, suspend, or reassign, the workflow service ensures that the requested action is contained in the list determined by the preceding algorithm.

Step 2 in the preceding algorithm deals with many cases. If a task is in a final, completed state (after all approvals in a sequential flow), an expired state, a withdrawn state, or an errored state, then no further update actions are permitted. In any of these states, the task, task history, and subtasks (parent task in parallel flow) can be viewed. If a task is suspended, then it can only be resumed or withdrawn. A task that is assigned to a group must be claimed before any actions can be performed on it.

Note: If you act on a task from the task details page, for example, if you approve a task, then any unchanged task details data is saved along with the saved changes to the task. However if you act on a task from the Actions menu, then unchanged task details are not saved.

29.4.1 System Actions

The action bar displays system actions, which are available on all tasks based on the user's privileges. [Table 29-7](#) lists system actions.

Table 29-7 System Task Actions

Action	Description
Claim	If a task is assigned to a group or multiple users, then the task must be claimed first. Claim is the only action available in the Task Action list for group or multiuser assignments. After a task is claimed, all applicable actions are listed.
Escalate	If you are not able to complete a task, you can escalate it and add an optional comment in the Comments area. The task is reassigned to your manager (up one level in a hierarchy).
Pushback	Use this action to send a task down one level in the workflow to the previous assignee. The pushback action overrides all other actions. For example, if a task is pushed back and then reassigned, after the reassignee approves it, the task goes to the user who performed the pushback. This is the expected behavior. Note: If the task is aggregated, then the Pushback action is not available.
Reassign	If you are a manager, you can delegate a task to reportees.

Table 29–7 (Cont.) System Task Actions

Action	Description
Release	If a task is assigned to a group or multiple users, it can be released if the user who claimed the task cannot complete the task. Any of the other assignees can claim and complete the task.
Renew	If a task is about to expire, you can renew it and add an optional comment in the Comments area. The task expiration date is extended one week. A renewal appears in the task history. The renewal duration for a task can be controlled by an optional parameter. The default value is P7D (seven days).
Submit Information and Request Information	Use these actions if another user requests that you supply more information or to request more information from the task creator or any of the previous assignees. If reapproval is not required, then the task is assigned to the next approver or the next step in the business process.
Suspend and Resume	If a task is not relevant, you can suspend it. These options are available only to users who have been granted the BPMWorkflowSuspend role. Other users can access the task by selecting Previous in the task filter or by looking up tasks in the Suspended status. A suspension is indefinite. It does not expire until Resume is used to resume working on the task.
Withdraw	If you are the creator of a task and do not want to continue with it, for example, you want to cancel a vacation request, you can withdraw it and add an optional comment in the Comments area. The business process determines what happens next. You can use the Withdraw action on the home page by using the Creator task filter.

29.4.2 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
- Re-initiate task
- Update outcome of task
- Completion of task
- Error of task
- Expiration of task
- Withdrawal of task
- Alerting of task to the error assignee

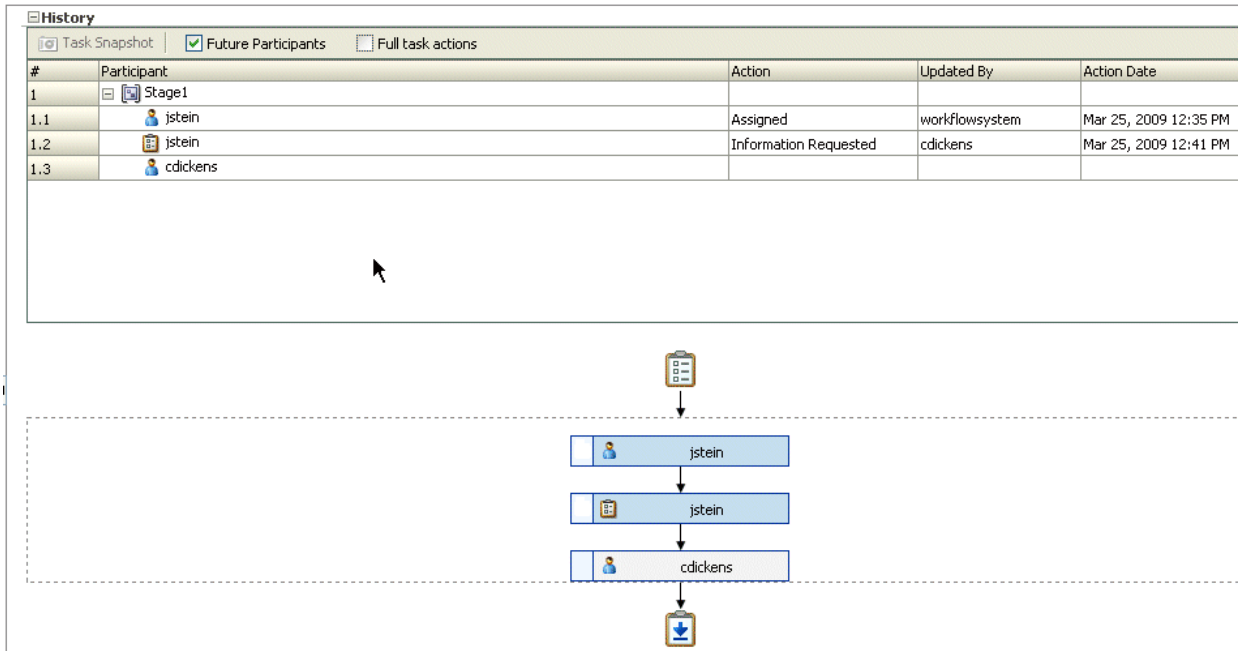
You can include the following actions in the short history list by modifying the `shortHistoryActions` element.

- Acquire
- Ad hoc route
- Auto release of task
- Delegate
- Escalate
- Information request on task
- Information submit for task
- Override routing slip

- Update outcome and route
- Push back
- Reassign
- Release
- Renew
- Resume
- Skip current assignment
- Suspend
- Update

The history provides a graphical view of a task flow, as shown in [Figure 29–20](#).

Figure 29–20 History: Graphical View



Check **Full task actions** to see all actions performed, including those that do not make changes to the task, such as adding comments, as shown in [Figure 29–21](#).

Figure 29–21 History: Full Task Actions

History

☐ Task Snapshot ☒ Future Participants ☒ Full task actions

#	Participant	Action	Updated By	Action Date
1	Stage 1			
1.1	jstein	Assigned	workflowsystem	Mar 25, 2009 12:35 PM
1.2	cdickens	Reassigned	jstein	Mar 25, 2009 12:40 PM
1.3	cdickens	Comment Added	cdickens	Mar 25, 2009 12:41 PM
1.4	jstein	Information Requested	cdickens	Mar 25, 2009 12:41 PM
1.5	cdickens			


```

graph TD
    Start((Task Icon)) --> J1[jstein]
    J1 --> C1[cdickens]
    C1 --> C2[cdickens]
    C2 --> J2[jstein]
    J2 --> C3[cdickens]
  
```

Available ways to view the task history include:

- Take a task snapshot
- See future approvers
- See complete task actions

29.4.3 How To Act on Tasks

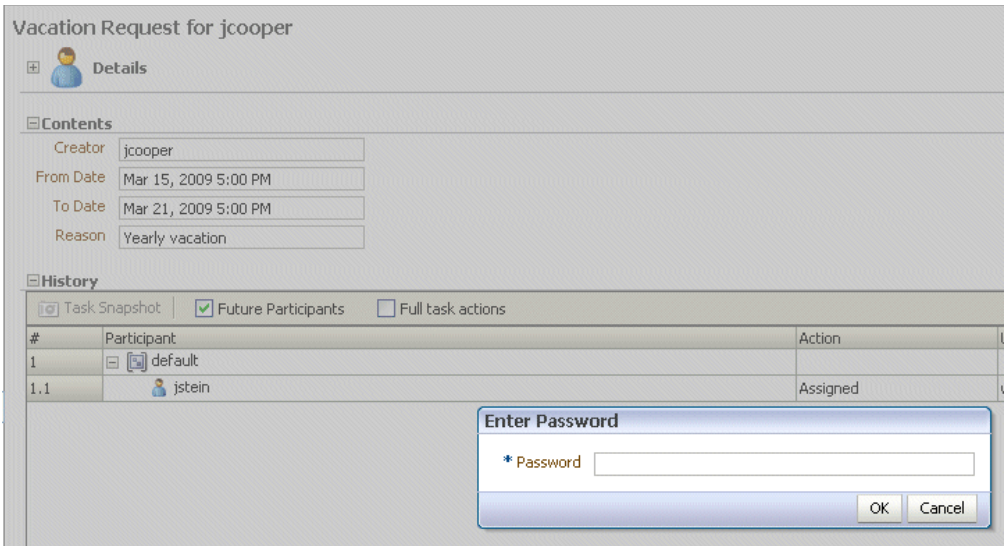
If the human task was designed to permit ad hoc routing, or if no predetermined sequence of approvers was defined, then the task can be routed in an ad hoc fashion in the worklist. For such tasks, a **Route** button appears on the task details page. From the Route page, you can look up one or more users for routing. When you specify multiple assignees, you can select whether the list of assignees is for simple (group assignment to all users), sequential, or parallel assignment.

Parallel tasks are created when a parallel flow pattern is specified for scenarios such as voting. In this pattern, the parallel tasks have a common parent. The parent task is visible to a user only if the user is an assignee or an owner or creator of the task. The parallel tasks themselves (referred to as subtasks) are visible to whomever the task is assigned, just like any other task. It is possible to view the subtasks from a parent task. In such a scenario, the task details page of the parent task contains a **View SubTasks** button. The SubTasks page lists the corresponding parallel tasks. In a voting scenario, if any of the assignees updates the payload or comments or attachments, the changes are visible only to the assignee of that task.

A user who can view the parent task (such as the final reviewer of a parallel flow pattern), can drill down to the subtasks and view the updates made to the subtasks by the participants in the parallel flow. The parent task is a container for the subtasks while they are worked on by the assignees. The task owner must not act on or approve the parent task.

If a human task was set up to require a password, then when you act on it, you must provide the password, as shown in [Figure 29–22](#).

Figure 29–22 Acting on a Task That Requires a Password

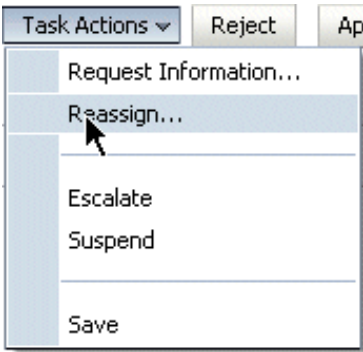


Note: Any kind of change to the task details page, such as changing a priority or adding a comment, requires you to save the change. If you add an attachment to a task, it is automatically saved.

To reassign or delegate a task:

1. From the **Task Actions** list, select **Reassign**, as shown in [Figure 29–23](#).

Figure 29–23 Reassigning a Task



2. Select **Reassign** or **Delegate**.

Delegate differs from **Reassign** in that the privileges of the delegatee are based on the delegator's privileges. This function can be used by managers' assistants, for example.

3. Provide or browse for a user or group name, as shown in [Figure 29–24](#).

Figure 29–24 Reassigning a Task

Reassign Task

Reassign tasks to one or more users.
This will transfer ownership of the task and remove it from your worklist.

☒ Reassign (transfer task to another user or group)
☐ Delegate (allow specified user to act on my behalf)

Users ▼ Search Reset
First name, Last name, ID

Available

- ☐ jausten
- ☐ jstein
- ☐ jverne

Selected

- ☒ jcooper
- ☒ jlondon

Move

Move All

Remove

Remove All

Details

Name	Jack London	Manager	sfitzger
Work Phone	100000008	Reportees	
Cell Phone	300000008	Roles	QATeam
Fax		Groups	LoanAgentGroup
Email	jlondon@emailExample.com		CentralRegion
Title	Loan Agent 1		

OK Cancel

A supervisor can always reassign tasks to any of his reportees.

4. Move names to the **Selected** area and click **OK**.

You can reassign to multiple users or groups. One of the assignees must claim the task, as shown in [Figure 29–25](#).

Figure 29–25 Claiming a Task

Vacation Request for jcooper

Details

Contents

Creator
 From Date
 To Date
 Reason

History

☒ Task Snapshot ☒ Future Participants ☐ Full task actions				
#	Participant	Action	Updated By	Action Date
1	default			
1.1	jstein	Assigned	workflowsystem	Mar 16, 2009 2:16 PM
1.2	jcooper, jlondon	Reassigned	jstein	Mar 16, 2009 2:58 PM

To request information:

1. From the **Task Actions** list, select **Request Information**, as shown in [Figure 29–26](#).

Figure 29–26 Requesting Information

Title	Number	Priority	Assignees	State	Created	Expires
Vacation Request for jcooper	200203	3	jstein (U)	Assigned	Mar 16, 2009 2:16 PM	

Vacation Request for jcooper

Details

Assignees: Creator, Expiration date: Acquired By, State: Assigned

Created: Mar 16, 2009 2:16 PM, Task Number: 200203, Priority: 2

Updated: Mar 16, 2009 2:28 PM

Task Actions ▼ | Reject

- Request Information...
- Reassign...
- Escalate
- Suspend
- Save

2. Request information from a past approver or search for a user name, or push the task back to the previous assignee, as shown in [Figure 29–27](#).

Figure 29–27 Requesting Information from Past Approvers or Another User, or Pushing the Task Back

Request More Information

☒ Request info from
 ☐ Push task back to previous assignee and then to me

☒ PastApprovers

Comments:

☒ Reapproval needed (retrace approval chain if applicable)

OK Cancel

If you use the **Search** icon to find a user name, the Identity Browser appears, as shown in [Figure 29–28](#).

Figure 29–28 Identity Browser

Identity Browser

Users Search Reset

First name, Last name, ID

Searched Items

jcooper

Details

Name	James Cooper	Manager	jstein
Work Phone	100000006	Reportees	
Cell Phone	300000006	Roles	
Fax		Groups	California
Email	jcooper@emailExample.com		Supervisor
Title	Loan Agent 1		LoanAgentGroup

OK Cancel

Note: If you are in a multi-tenancy environment, search for a user simply by the user identifier and not by the tenant identifier. For example, if the user identifier is jstein and the tenant identifier is company_name.jstein, you would search by using jstein.

3. Click **OK**.

To route a task:

1. From the **Task Actions** list, select **Adhoc Route**, as shown in [Figure 29–29](#).

Figure 29–29 Ad Hoc Routing

Help Desk request for wfaulk

Task Actions RESOLVED

Request Information...
Reassign...
Adhoc Route...
Escalate
Suspend
Renew
Save

Details

Contents

Location California

Type Hardware

Problem Description Unable to reboot the system

Severity 2

Status Created

Requester

ID wfaulk

First Name William

Last Name Faulkner

2. Select an action and a routing option, as shown in [Figure 29–30](#).

Figure 29–30 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, then one member must claim the task and act on it.
- **Group Vote:** Use this option when multiple users, working in parallel, must act, such as in a hiring situation when multiple users vote to hire or reject an applicant. You specify the voting percentage that is needed for the outcome to take effect, such as a majority vote or a unanimous vote, as shown in [Figure 29–31](#).

Figure 29–31 Providing Consensus Information

Default outcome Reject

Consensus Percentage 80

Minimum agreement to override Default

- **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. Add optional comments for the next participant on the route.
 4. Provide or search for user or group names; then move the names to the **Selected** area.
 5. Click **OK**.

To add comments:

Note: Comments cannot be deleted once they are added.

1. In the **Comments** area of the Worklist details panel, click **Add**. The Add Comment dialog box appears.

Figure 29–32 Adding Comments in Worklist Application

Add Comment

* Comment

Upload to Process ☐ (If unchecked, uploaded by default to Task)

OK Cancel

Comments

No data to display

2. Enter your comment text.
3. To attach your comment to the process rather than to this task, select **Upload to Process**. To attach the comment to the task, leave the check box unselected.
4. Click **OK**. The date and timestamp and your user name are included with the comment.

Note: Attachment file names that use a multibyte character set (MBCS) are not supported.

Attachments of up to 1998K can be uploaded. You can modify this setting by setting the context parameter in `web.xml` as follows:

```
<context-param>
  <param-name>org.apache.myfaces.trinidad.UPLOAD_MAX_DISK_
SPACE</param-name>
  <param-value>1998</param-value>
</context-param>
```

For more information about file uploading, see the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application*.

5. From the Task Actions list, click **Save**.

To add attachments:

Notes:

- Before you browse for or upload attachments, click **Save** to ensure that any previous changes to the task details page are saved.
- When you remove a file or URL attachment, the task is not automatically updated. You must explicitly select **Actions > Save**. Otherwise, the attachment is not removed, even though it is displayed as removed. This is the expected behavior.
- If you add a file attachment, you do not need to explicitly select **Actions > Save**.
- If you add a URL attachment, you must explicitly select **Actions > Save**.
- Attachment file names that use a multibyte character set (MBCS) are not supported.

Attachments of up to 1998K can be uploaded. You can modify this setting by setting the context parameter in `web.xml` as follows:

```
<context-param>
  <param-name>org.apache.myfaces.trinidad.UPLOAD_MAX_DISK_
SPACE</param-name>
  <param-value>1998</param-value>
</context-param>
```

For more information about file uploading, see the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application*.

- If you attach a URL file in Oracle BPM Worklist (for example, `http://www.oracle.com/technology/products/oem/management_partners/snmpwp6.gif`), it is not sent as an e-mail attachment. Instead, it appears as a hyperlink in the task details of the e-mail notification. However, if a desktop file is attached, it can be seen as a separate attachment in the task notification.
-

1. In the Attachments area of the Worklist details panel, click **Add Attachment**. The Add Attachment dialog box appears as shown the example in [Figure 29-33](#).

Figure 29–33 Adding a Worklist Attachment

Note: Make sure that all previous updates to task including url attachments are saved before uploading file.

Attachment Type: ☐ Desktop File ☐ URL ☒ UCM Document

* Name: InsurancePolicy.doc

* URL: c:\claims\InsurancePolicy.doc

Parameter	Value
Department	Finance
Insurance Amount	300.00
Claim Number *	93442567

Upload to Process ☐ (If unchecked, uploaded to Task by default)

OK Cancel

2. Specify the attachment type. Options are:

- Desktop File
- URL
- UCM document

If you are specifying a document stored in Oracle Universal Content Management:

- a. Select **UCM document**. The dialog box changes to corresponds to your selection.
- b. Enter the name and URL of the document stored in Oracle Universal Content Management.
- c.
- d.
- e. Click **OK**.

29.4.4 How To Act on Tasks That Require a Digital Signature

The worklist supports the signature policy created in the human task:

- **No signature required** — Participants can send and act on tasks without providing a signature.
- **Password required** — Participants must specify their login passwords.
- **Digital certificate (signature) required** —Participants must possess a digital certificate before being able to send and act on tasks. A digital certificate contains the digital signature of the certificate-issuing authority so that anyone can verify that the certificate is real. A digital certificate establishes the participant's

credentials. It is issued by a certification authority (CA). It contains your name, a serial number, expiration dates, a copy of the certificate holder's public key (used for encrypting messages and digital signatures), and the digital signature of the certificate-issuing authority so that a recipient can verify that the certificate is real.

When you act on a task that has a signature policy, the **Sign** button appears, as shown in [Figure 29–34](#).

Figure 29–34 Digital Signature Task Details

Digital Signature Task Details

Sign

Back To History

Approve Order

Task Number: 200002

State: Assigned

Outcome

Priority: 3

Creator jstein

Created Date: 5/15/09 6:56:47 AM

Updated Date: 5/15/09 6:56:47 AM

Expiration Date:

Assignees: jcooper

Acquired By:

Customer Information

CustID 10

ID 23

Street

City

State

Zip

Country

Next

The evidence store service is used for digital signature storage and nonrepudiation of digitally signed human tasks. You can search the evidence store, as shown in [Figure 29–35](#).

Figure 29–35 The Evidence Store

Search Evidence Store

Match ☒ All ☐ Any

Signer Signed After Signed Before

Advanced

Signature Policy Verified Date Task Outcome

Status Task Number

Signature Policy	Signer	Status	Creation Date	Signed Date

See [Section 31.1.10, "Evidence Store Service and Digital Signatures"](#) for more information.

To provide a digital signature:

1. In the upper right corner of Oracle BPM Worklist, click **Preferences**.
2. In the navigation bar on the left, click **Certificates**.
3. Upload the certificate to use to sign your decision, as shown in [Figure 29–36](#).

When signing a task outcome using your certificate, you must upload the entire chain of certificates through Oracle BPM Worklist as a .P7B (PKCS7 format) file, not just the one certificate issued to you by the certificate issuer. The entire chain can be exported through Internet Explorer. Mozilla Firefox does not let you export the chain as a .P7B file. Therefore, you can perform the following steps:

- a. Export the chain from Mozilla Firefox as a .P12 file (PKCS12 format that also contains your private key).
- b. Import the .P12 file in Internet Explorer.
- c. Export it again from Internet Explorer as a .P7B file.
- d. Upload it through Oracle BPM Worklist.

Figure 29–36 Uploading a Certificate

Note the following important points when providing your certificate to the system. Otherwise, you cannot use your certificate to sign your decisions on tasks.

- The PKCS7 file format is a binary certificate format. Select this option if you have a standalone certificate file stored on your disk.
 - The PKCS12 file format is a keystore format. Select this option if you have your certificate stored inside a keystore.
 - If you want to copy and paste the contents of the certificate, select **Type or Paste Certificate Contents** and paste the BASE64-encoded text into the field. Do not paste a certificate in any other format into this field. Likewise, if you choose to upload a certificate, do not try to upload a BASE64-encoded certificate. Only PKCS12 and PKCS7 formatted files are supported for uploads.
4. Return to the task list by clicking the **Home** link in the upper-right corner of Oracle BPM Worklist.
 5. Click a task to approve or reject.
The task details are displayed.
 6. Click either **Approve** or **Reject**.
Details about the digital signature are displayed.
 7. For a task that has a signature policy, click **Sign**.
The Text Signing Report dialog appears.
 8. Select the certificate from the dropdown list to use to sign your decision.
 9. Enter the master password of the web browser that you are using.
 10. Click **OK**.

The web browser signs the string displayed in the upper half of the Text Signing Request with the certificate you selected and invokes the action (approval or rejection) that you selected. The task status is appropriately updated in the human workflow service.

For more information about how certificates are uploaded and used, see [Section 31.1.10, "Evidence Store Service and Digital Signatures."](#)

29.5 Approving Tasks

[Table 29–8](#) describes the type of actions that can be performed on tasks by the various task approvers.

Table 29–8 Task Actions and Approvers

Task Action	Admin	Owner (+ Owner Group)	Assignee (+ Assignee Manager + Assignee Group + Proxy Assignee)	Creator	Reviewer	Approver
Acquire (Claim)	No	Yes	Yes	No	No	No
Custom	No	Yes ¹	Yes ¹	No	No	No
Delegate	No	No	Yes	No	No	No
Delete	No ²	No ²	Yes ²	Yes ²	No	No
Error	No	No	Yes ³	No	No	No
Escalate	Yes ⁴	Yes ⁴	Yes	No	No	No
Info Request	No	No	Yes	No	No	No
Info Submit	No	No	Yes	No	No	No
Override Routing Slip	Yes	Yes	No	No	No	No
Push Back	No	No	Yes	No	No	No
Purge	Yes ²	Yes ²	No	Yes	No	No
Reassign	Yes ⁵	Yes ⁵	Yes (No for proxy assignee)	No	No	No
Release	Yes	Yes	Yes	No	No	No
Renew	No	Yes	Yes	No	No	No
Resume	Yes	Yes	Yes	No	No	No
Route	No	Yes	Yes	No	No	No
Skip Current Assignment	Yes	Yes	No	No	No	No
Suspend	Yes	Yes	Yes	No	No	No
Update	No	Yes	Yes	Yes	No	No
Update Attachment	Yes	Yes	Yes	Yes	Yes	No
Update Comment	Yes	Yes	Yes	Yes	Yes	No
View Process History	Yes	Yes	Yes	Yes	No	No

Table 29–8 (Cont.) Task Actions and Approvers

Task Action	Admin	Owner (+ Owner Group)	Assignee (+ Assignee Manager + Assignee Group + Proxy Assignee)	Creator	Reviewer	Approver
View Sub Tasks	Yes	Yes	Yes	No	No	No
View Task History	Yes	Yes	Yes	Yes	Yes	Yes
Withdraw	Yes	Yes	No	Yes	No	No

¹ Not valid for ToDo tasks

² Valid only for ToDo tasks

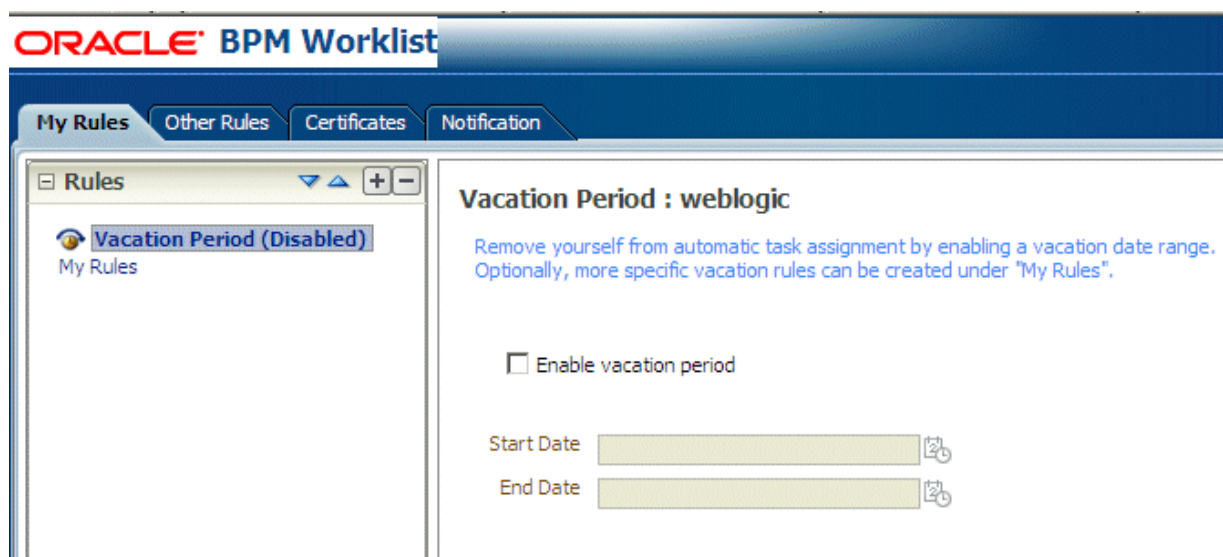
³ Applicable for tasks in alerted states

⁴ Without claim and escalate to current assignee's manager

⁵ Without claim

29.6 Setting a Vacation Period

You can set a vacation period so that you are removed from automatic task assignment during the dates you specify, as shown in [Figure 29–37](#).

Figure 29–37 Setting a Vacation Period

Vacation rules are not executed for ToDo tasks. See [Section 29.7, "Setting Rules,"](#) for how to set a vacation rule that is synchronized with the vacation period.

To create a vacation period:

1. Click the **Preferences** link.
The **My Rules** tab is displayed.
2. Click **Enable vacation period**.
3. Provide start and end dates.
4. Click **Save**.

The vacation period is enabled, as shown in [Figure 29–38](#).

Figure 29–38 Enabling a Vacation Period

The screenshot shows the Oracle BPM Worklist interface. The 'My Rules' tab is selected. In the left sidebar, under 'Rules', the 'Vacation Period (Enabled)' rule is listed under 'My Rules'. The main content area is titled 'Vacation Period' and contains the following text: 'Remove yourself from automatic task assignment by enabling a vacation date range. Optionally, more specific vacation rules can be created under 'My Rules'.'

Below the text, there is a checkbox labeled 'Enable vacation period' which is checked. Underneath, there are two date input fields: 'Start Date' with the value 'Mar 21, 2009' and 'End Date' with the value 'Mar 27, 2009'. Each date field has a small calendar icon to its right.

29.7 Setting Rules

Rules act on tasks, either a specific task type or all the tasks assigned to a user or group. [Figure 29–39](#) shows where you set rules, including vacation rules (different from the vacation period settings described in [Section 29.6, "Setting a Vacation Period"](#)).

Figure 29–39 Creating a Rule

The screenshot shows the Oracle BPM Worklist interface for creating a new rule. The 'My Rules' tab is selected. In the left sidebar, under 'Rules', the 'User Rule' is listed under 'My Rules'. The main content area is titled 'My Rule' and contains the following fields and options:

- Name:** A text input field containing 'User Rule'.
- Use as vacation rule:** A checkbox that is currently unchecked.
- Apply only to task type(s):** A text input field with a search icon to its right.
- Execute rule only between these dates:** A checkbox that is currently unchecked.
- Start Date:** A date input field.
- End Date:** A date input field.
- IF:** A section for defining conditions. It includes an 'Add Condition' button and a dropdown menu currently showing 'Start Date'.
- THEN:** A section for defining actions. It includes four radio button options:
 - Reassign to:** A radio button option with a dropdown menu showing 'User' and a search icon.
 - Delegate to:** A radio button option with a search icon.
 - Set outcome to:** A radio button option with a dropdown menu.
 - Take no action:** A radio button option.

Below the 'THEN' section, there is a small text block: 'Reassigned task access is determined according to new assignee rights. Delegated task access is determined according to rights of original user who delegates. 'Take no action' is used to create exception rules that override a more generic rule.'

A rule cannot always apply in all circumstances in which it is used. For example, if a rule applies to multiple task types, it may not be possible to set the outcome for all tasks, since different tasks can have different outcomes.

Rules are executed in the order in which they are listed. Rules can be reordered by using the up and down buttons in the header, as shown in [Figure 29–39](#).

If a rule meets its filter conditions, then it is executed and no other rules are evaluated. For your rule to execute, you must be the only user assigned to that task. If the task is assigned to multiple users (including you), the rule does not execute.

You cannot specify business rules for ToDo tasks

29.7.1 How To Create User Rules

Specify the following when creating a user rule:

- Rule name
- If the rule is a vacation rule. See [Section 29.6, "Setting a Vacation Period,"](#) for how to set the vacation period that is synchronized with the vacation rule.
- Which task or task type the rule applies to—If unspecified, then the rule applies to all tasks. If a task type is specified, then any attributes mapped for that task type can be used in the rule condition.
- When the rule applies
- Conditions on the rule—These are filters that further define the rule, such as specifying that a rule acts on priority 1 tasks only, or that a rule acts on tasks created by a specific user. The conditions can be based on standard task attributes and any mapped attributes that have been mapped for the specific tasks. See [Section 29.10.1, "How To Map Attributes,"](#) for more information.

User rules do the following actions:

- **Reassign to**—You can reassign tasks to subordinates or groups you manage.
- **Delegate to**—You can delegate to any user or group. Any access rights or privileges for completing the task are determined according to the original user who delegated the task. (Any subsequent delegations or re-assignments do not change this from the original delegating user.)
- **Set outcome to**—You can specify an automatic outcome if the workflow task was designed for those outcomes, for example, accepting or rejecting the task. The rule must be for a specific task type. If a rule is for all task types, then this option is not displayed.
- **Take no action**—Use this action to prevent other more general rules from applying. For example, to reassign all your tasks to another user while you are on vacation, except for loan requests, for which you want no action taken, then create two rules. The first rule specifies that no action is taken for loan requests; the second rule specifies that all tasks are reassigned to another user. The first rule prevents reassignment for loan requests.

To create a user rule:

1. Click the **Preferences** link
The **My Rules** tab is displayed.
2. In the **Rules** area, click **My Rules** and click **Add**.
3. In the **My Rule** area, do the following and click **Save**:

- Provide a name for the rule.
- Select **Use as a vacation rule** if you are creating a vacation rule. The start and end dates of the rule are automatically synchronized with the vacation period.
- Browse for task types to which the rule applies.
- Select **Execute rule only between these dates** and provide rule execution dates.
- In the **IF** area, add rule conditions.
- In the **THEN** area, select actions to be taken: **Reassign to**, **Delegate to**, **Set outcome to**, or **Take no action**), as shown in [Figure 29–39](#).

The new rule appears under the **My Rules** node.

29.7.2 How To Create Group Rules

Creating a group rule is similar to creating a user rule, with the addition of a list of the groups that you (as the logged-in user) manage. Examples of group rules include:

- Assigning tasks from a particular customer to a member of the group
- Ensuring an even distribution of task assignments to members of a group by using round-robin assignment
- Ensuring that high-priority tasks are routed to the least busy member of a group

Group rules do the following actions:

- **Assign to member via**—You can specify a criterion to determine which member of the group gets the assignment. This dynamic assignment criterion can include round-robin assignment, assignment to the least busy group member, or assignment to the most productive group member. You can also add your custom functions for allocating tasks to users in a group.
- **Assign to**—As with user rules, you can assign tasks to subordinates or groups you directly manage.
- **Take no action**—As with user rules, you can create a rule with a condition that prevents a more generic rule from being executed.

To create a group rule:

1. Click the **Preferences** link
2. Click the **Other Rules** tab.
3. Select **Group** from the list.
4. Enter a group name and click the **Search** icon, or enter a group name.

The Identity Browser opens for you to find and select a group.

5. Select the group name under the **Group Rules** node and click **Add**, as shown in [Figure 29–40](#).

Figure 29–40 Creating a Group Rule

6. Provide group rule information and click **Save**.
 - Provide a name for the rule.
 - Browse for task types to which the rule applies.
 - Provide rule execution dates.
 - In the **IF** area, add rule conditions.
 - In the **THEN** area, select the actions to be taken (or none) (**Assign to member via**, **Assign to**, or **Take no action**), as shown in [Figure 29–40](#).

The new rule appears under the **Group Rules** node.

29.7.3 Assignment Rules for Tasks with Multiple Assignees

If a task has multiple assignees, then assignment rules are not evaluated for the task, and the task is not automatically routed. This is because each of the task's assignees can define assignment rules, which can potentially provide conflicting actions to take on the task. Only tasks that are assigned exclusively to a single user are routed by the assignment rules.

For example, consider the following sequence:

1. A rule is created for user cdickens to reassign all assigned requests to user jstein.
2. User jcooper reassigns the allocated tasks to cdickens and cdoyle.
3. Cdickens claims the task, and the task appears in their inbox.

The task is not automatically reassigned to jstein. The task is routed to jstein, following the assignment rule set for cdickens, if user jcooper explicitly re-assigns the task only to cdickens instead of reassigning the task to multiple users (cdickens and cdoyle).

29.8 Using the Worklist Administration Functions

Administrators are users who have been granted the BPMWorkflowAdmin role. Administration functions include the following:

- Managing other users' or groups' rules
- Setting the worklist display (application preferences). Application preferences customize the appearance of the worklist, including:
 - The login realm label
 - The resource bundle
 - Where the language locale information is retrieved from
 - The branding logo
 - The branding title
 - The branding skin
 - Any external applications you want to use, for example, `oracle.com`, or `google.com`
- Specifying mapped attributes

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.

This section contains these topics:

- [Section 29.8.1, "How To Manage Other Users' or Groups' Rules \(as an Administrator\)"](#)
- [Section 29.8.2, "How to Specify the Login Page Realm Label"](#)
- [Section 29.8.3, "How to Specify the Resource Bundle"](#)
- [Section 29.8.4, "How to Specify the Language Locale Information"](#)
- [Section 29.8.5, "How to Specify a Branding Logo"](#)
- [Section 29.8.6, "How to Specify the Branding Title"](#)
- [Section 29.8.7, "How to Choose a Skin"](#)
- [Section 29.8.8, "How to Enable Customized Applications and Links"](#)

For information about specifying mapped attributes, see [Section 29.10, "Using Mapped Attributes \(Flex Fields\)"](#)

29.8.1 How To Manage Other Users' or Groups' Rules (as an Administrator)

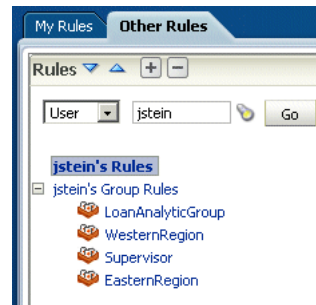
This function is useful for fixing a problem with a rule. Also, for a user who no longer works for the company, administrators can set up a rule for that user so that all tasks assigned to the user are automatically assigned to another user or group.

To create a rule for another user or group:

1. From the task list page, click the **Rules** link.
2. Click the **Other Rules** tab.

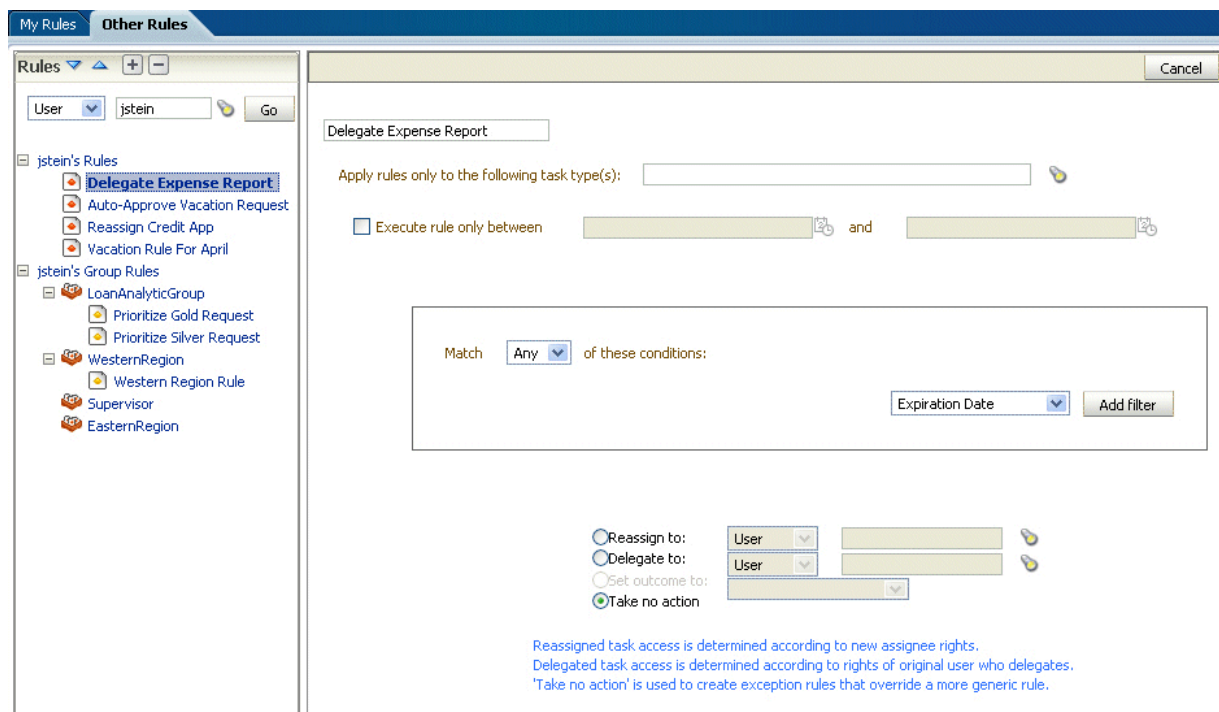
3. Search for the user or group for whom rules are to be created, as shown in [Figure 29–41](#).

Figure 29–41 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 29–42](#), and click **Save**.

Figure 29–42 Defining Rules for Another User or Group



29.8.2 How to Specify the Login Page Realm Label

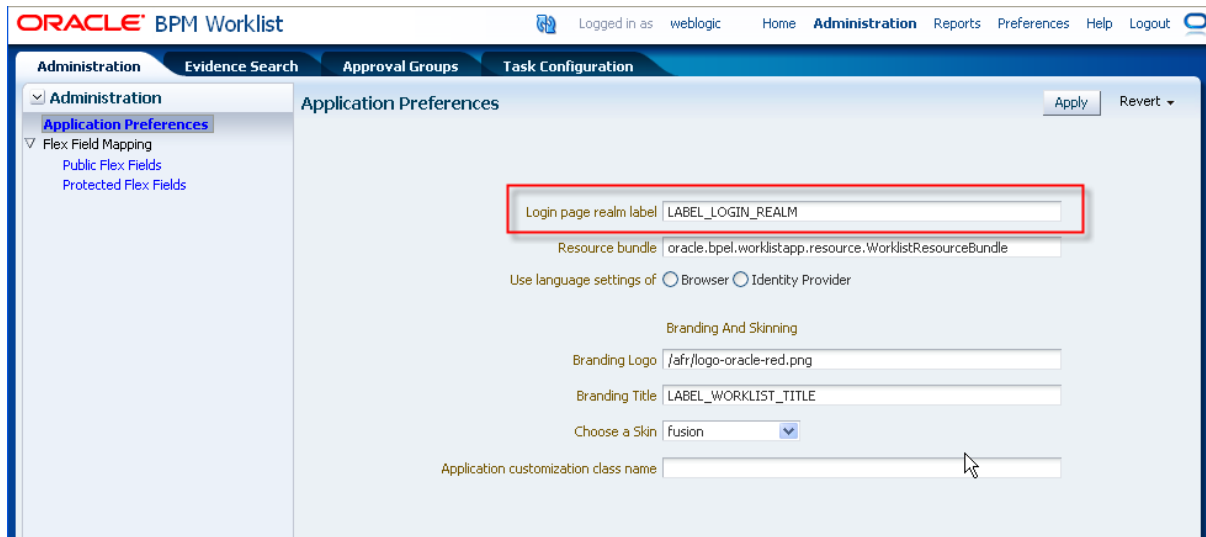
If the identity service is configured with multiple realms, then, when a user logs in to Oracle BPM Worklist, the login page displays a list of realm names. LABEL_LOGIN_REALM specifies the resource bundle key used to look up the label to display these realms.

You can change the term *realm* to fit the user community—terms such as *country*, *company*, *division*, or *department* may be more appropriate. To change the term *realm*,

customize the resource bundle, specify a resource bundle key for this string, and then set the **Login page realm label** parameter to point to that resource bundle key.

Figure 29–43 shows the Application Preferences page with the **Login page realm label** field highlighted. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 29–43 Specifying the Login Page Realm Label



29.8.3 How to Specify the Resource Bundle

The resource bundle provides the strings displayed in the Worklist Application. By default, the class path to the resource bundle is:

```
oracle.bpel.worklistapp.resource.WorklistResourceBundle
```

Figure 29–44 shows the Application Preferences page with the **Resource Bundle** field highlighted. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 29–44 Specifying the Resource Bundle

The screenshot shows the Oracle BPM Worklist Administration interface. The left sidebar has a tree view with 'Administration' expanded, showing 'Application Preferences', 'Flex Field Mapping', 'Public Flex Fields', and 'Protected Flex Fields'. The main content area is titled 'Application Preferences' and contains several configuration fields. A red rectangular box highlights the 'Resource bundle' field, which contains the text 'oracle.bpel.worklistapp.resource.WorklistResourceBundle'. Other visible fields include 'Login page realm label' (LABEL_LOGIN_REALM), 'Use language settings of' (radio buttons for Browser and Identity Provider), 'Branding And Skinning' section with 'Branding Logo' (/afr/logo-oracle-red.png), 'Branding Title' (LABEL_WORKLIST_TITLE), 'Choose a Skin' (fusion), and 'Application customization class name'.

As an administrator, you can add or modify strings shown in the application by creating a custom resource bundle. You can then use the **Resource Bundle** field in the Application Preferences page to specify the class path to your custom resource bundle.

For more information about customizing resource bundles, see *Oracle Fusion Middleware User's Guide for Oracle Business Process Management*.

29.8.4 How to Specify the Language Locale Information

From the Application Preferences page, you can specify how the Worklist Application display language is determined. Information about the language locale can be derived from either the user's browser or the identity provider that stores information on worklist users.

You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 29–45 Specifying Language Local Information

This screenshot is similar to Figure 29-44, showing the 'Application Preferences' page. In this instance, a red rectangular box highlights the 'Use language settings of' field, which has radio buttons for 'Browser' and 'Identity Provider'. The 'Resource bundle' field is also visible and contains the same text as in the previous figure. The rest of the page layout, including the sidebar and other configuration fields, is identical to Figure 29-44.

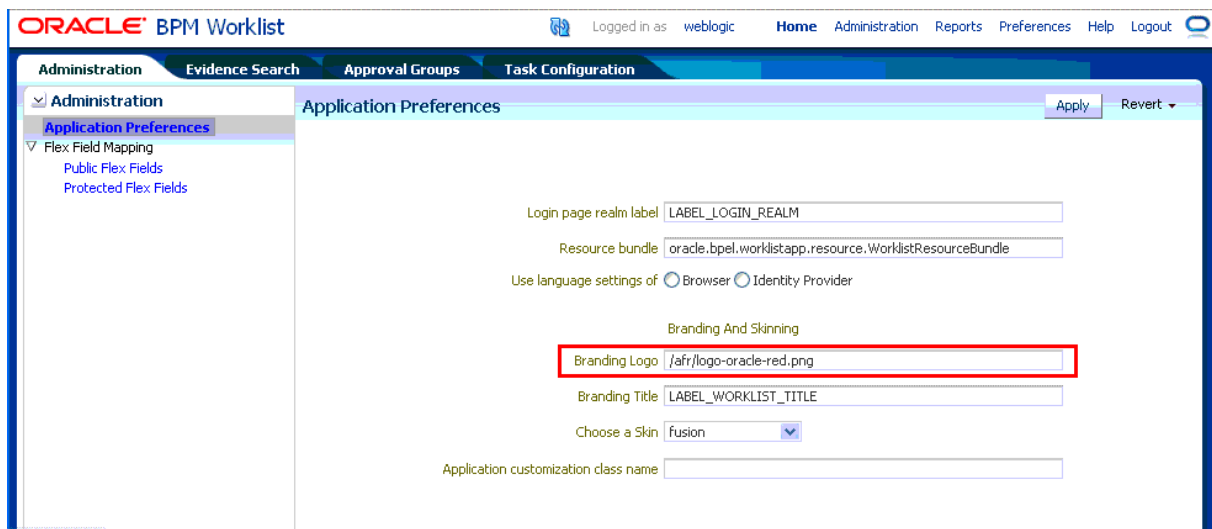
29.8.5 How to Specify a Branding Logo

A branding logo is the image displayed in the top left corner of every page of the Worklist Application. The Oracle logo is the default, and you can change it to one of your choosing.

Note: The ideal image size is 120px x 40px (length x width) for proper display. Although images with high resolution and size are compressed to fit the branding logo size, smaller images display better.

Figure 29–46 shows the Application Preferences page with the **Branding Logo** field highlighted. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 29–46 Specifying the Branding Logo



To specify the branding logo:

Do one of the following:

- Refer to an external image-hosting website. To do this: In the **Branding Logo** field, enter the URL of the image .
- Upload an image to a particular location on the server and, in the **Branding Logo** field, enter its relative path, for example, /afr/my_logo.png.
- Refer to an image from the shared library. To do this: In the **Branding Logo** field, enter the path of the logo name as found in the shared library, for example, /my_logo.pngv.

Note: Customizing the branding logo from either the Worklist Application or Process Workspace changes the logo in both applications. For example, if you change the logo from Worklist, the Workspace logo is changed automatically.

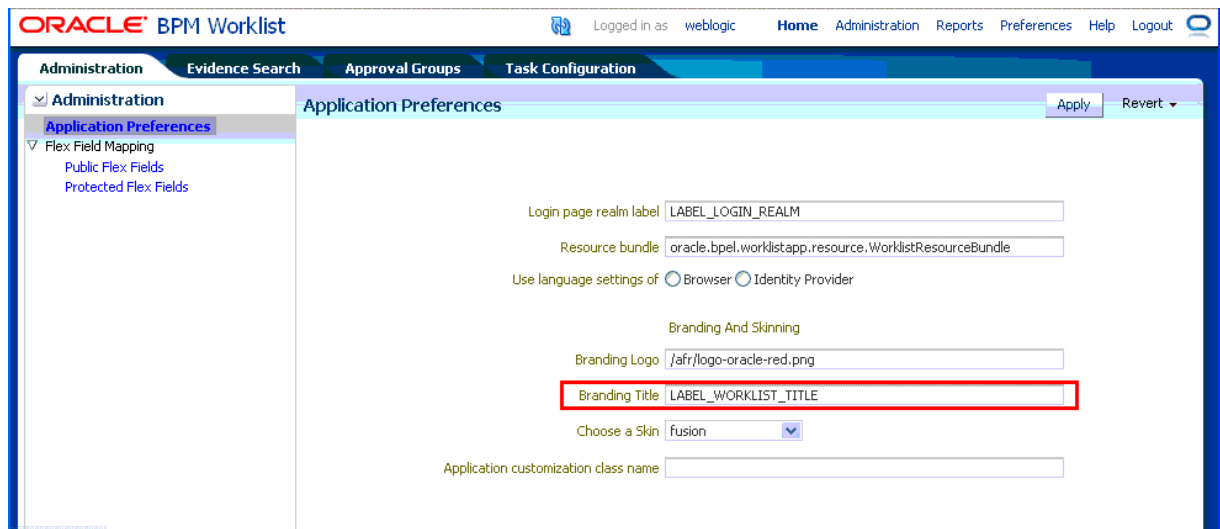
For information about deploying images and JAR files as part of a shared library, see *Oracle Fusion Middleware User's Guide for Oracle Business Process Management*.

29.8.6 How to Specify the Branding Title

You can specify the title for your site, changing the default title, *BPM Worklist*, to one that you choose.

Figure 29–47 shows the Application Preferences field with the **Branding Title** field highlighted. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 29–47 Specifying the Branding Title



To specify the branding title:

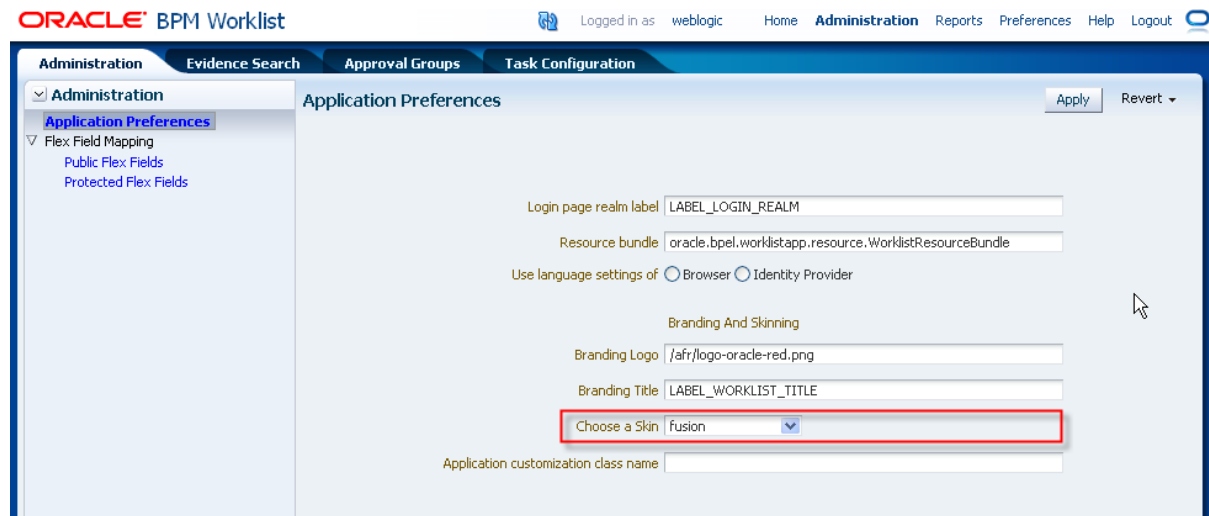
Do one of the following:

- In the **Branding Title** field, enter a simple string for your title.
- In the **Branding Title** field, enter a label that refers to a key-value pair in the Resource Bundle. In this way, you can internationalize your title, for example, LABEL_WORKLIST_TITLE.

29.8.7 How to Choose a Skin

A skin determines the look and feel of your graphical interface. You specify the skin from the Application Preferences page. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 29–48 shows the Application Preferences page with the Choose a Skin field highlighted.

Figure 29–48 Choosing a Skin**To choose a skin:**

Do one of the following:

- From the **Choose a Skin** list, select one of the default ADF skins
- Upload your own customized skin .css file in a .JAR file and deploy it as a part of shared library. Then, when you restart your application from the console, your custom skin appears in the **Choose a Skin** list.

To create a JAR file containing customized skins:

1. Create a directory structure similar to the following example:

```
C:\temp\META-INF\adf\oracle\skin\images
META-INF\skins\custom.css
META-INF\trinidad-skins.xml
```

Note that, in this example, you can change the word *custom* to the name of your own customized skin.

2. Make sure the content of trinidad-skins.xml file is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
  <skin>
    <id>custom.desktop</id>
    <family>custom</family>
    <extends>custom.desktop</extends>
    <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
    <style-sheet-name>skins/custom.css</style-sheet-name>
  </skin>
</skins>
```

3. Create the .JAR file by issuing the following command from the c:\temp directory:

```
jar -cvf customSkin.jar META-INF/
```

4. Copy this JAR file to the directory /scratch/username/sharedLib.

Note: Refer to the images in your css file this way:

```
../adf/oracle/skin/images/example.gif (with the two trailing dots).
```

This allows the search for the META-INF root to start one directory above the META-INF/skin directory in which the .css file is located.

For information about deploying JAR files as part of a shared library, see *Oracle Fusion Middleware User's Guide for Oracle Business Process Management*.

29.8.8 How to Enable Customized Applications and Links

For Process Workspace, you can create customized external applications and links that become available in the External Applications panel. Moreover, in both Process Workspace and the Worklist Application, you can specify the columns that appear in the Task Details pane.

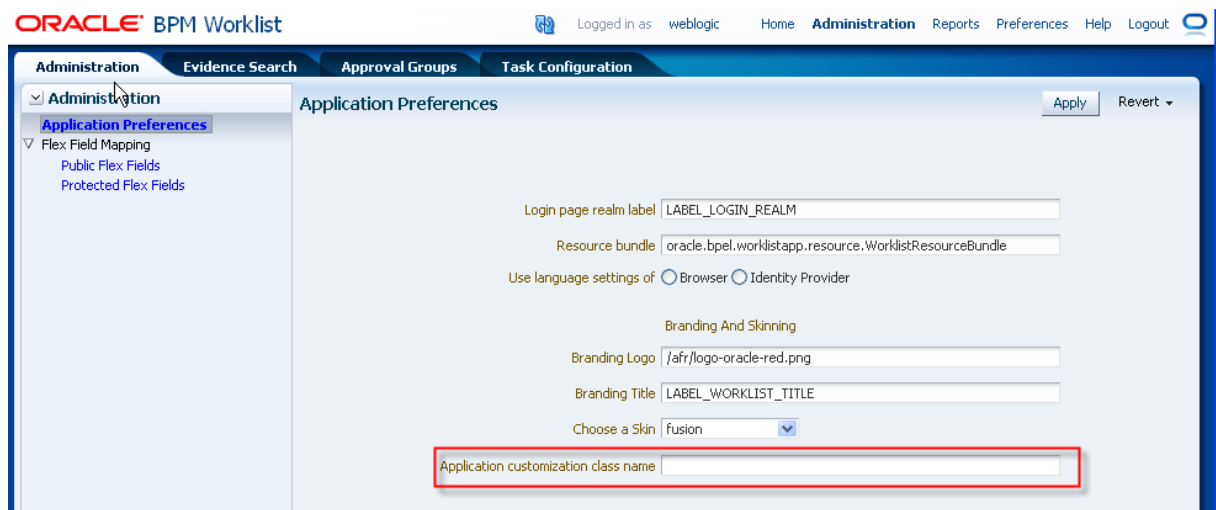
You specify a custom application by using the Application Preferences page. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

To see the Java code for specifying a custom application, see [Section 29.14, "Java Code for Enabling Customized Applications in Worklist Application"](#) on page 29-88.

To enable customized applications:

1. In the Application Preferences page, enter the class name of your custom application in the **Application customization class name** field as shown in [Figure 29-49](#).

Figure 29-49 Specifying a Custom Application

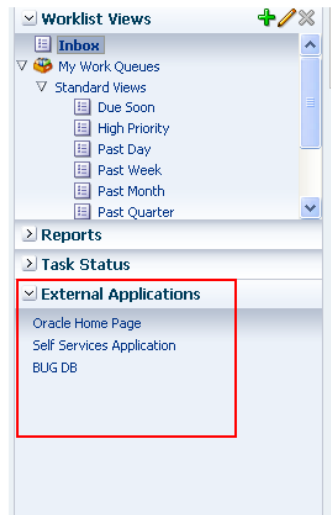


2. Restart the application from the console.
Depending on your customization, you can now see it's effects.

If your customization is for Process Workspace and involves either creating an external application or specifying inbox columns in the Task Details pane or both, you will see the following:

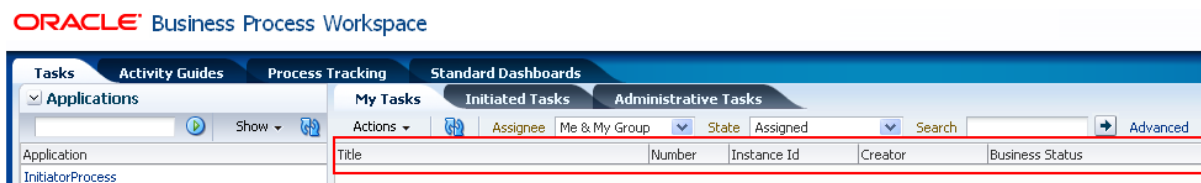
- Your custom application listed in the External Applications panel of the Process Workspace Home page as shown in [Figure 29–50](#).

Figure 29–50 External Applications Panel in Process Workspace



- The columns of the Task Details inbox adjusted according to your specifications as shown in [Figure 29–51](#).

Figure 29–51 Customized Columns in Task Details Pane



For more information about customizing applications in Worklist Application and Process Workspace, see *Oracle Fusion Middleware User's Guide for Oracle Business Process Management*.

29.9 Specifying Notification Settings

You can configure the notification settings to control how, when, and where you receive messages in cases when you have access to multiple communication channels (delivery types). Specifically, you can define messaging filters (delivery preferences) that specify the channel to which a message should be delivered, and under what circumstances.

For example, you might want to create filters for messages received from customers with different Service Level Agreements (SLA), specifying to be notified through business phone and SMS channels for customers with a premium SLA and by EMAIL for customers with a nonpremium SLA.

29.9.1 Messaging Filter Rules

A messaging filter rule consists of *rule conditions* and *rule actions*. A rule condition consists of a rule attribute, an operator, and an associated value. A rule action is the action to be taken if the specified conditions in a rule are true.

29.9.1.1 Data Types

[Table 29–9](#) lists data types supported by messaging filters. Each attribute has an associated data type, and each data type has a set of predefined comparison operators.

Table 29–9 *Data Types Supported by Messaging Filters*

Data Type	Comparison Operators
Date	isEqual, isNotEqual, isGreaterThan, isGreaterThanOrEqual, isLessThan, isLessThanOrEqual, Between, isWeekday, isWeekend
Time	isEqual, isNotEqual, Between
Number	isEqual, isNotEqual, Between, isGreaterThan, isGreaterThanOrEqual, isLessThan, isLessThanOrEqual
String	isEqual, isNotEqual, Contains, NotContains

Note: The String data type does not support regular expressions.

29.9.1.2 Attributes

[Table 29–10](#) lists the predefined attributes for messaging filters.

Table 29–10 *Predefined Attributes for Messaging Filters*

Attribute	Data Type
Total Cost	Number
From	String
Expense Type	String
To	String
Application Type	String
Duration	Number
Application	String
Process Type	String
Status	String
Subject	String
Customer Type	String
Time	Time
Group Name	String
Processing Time	Number
Date	Date
Due Date	Date
User	String

Table 29–10 (Cont.) Predefined Attributes for Messaging Filters

Attribute	Data Type
Source	String
Amount	Number
Role	String
Priority	String
Customer Name	String
Expiration Date	Date
Order Type	String
Organization	String
Classification	String
Service Request Type	String

29.9.2 Rule Actions

For a given rule, a messaging filter can define the following actions:

- **Send No Messages:** Do not send a message to any channel.
- **Send Messages to All Selected Channels:** Send a message to all specified channels in the address list.
- **Send to the First Available Channel:** Send a message serially to channels in the address list until one successful message is sent. This entails performing a send to the next channel when the current channel returns a failure status. This filter action is not supported for messages sent from the human workflow layer.

29.9.3 Managing Messaging Channels

In Oracle BPM Worklist, messaging channels represent both physical channels, such as business mobile phones, and also email client applications running on desktops. Specifically, Oracle BPM Worklist supports the following messaging channels:

- EMAIL
- IM
- MOBILE
- SMS
- VOICE
- WORKLIST

Note the following about message channels:

- Addresses for messaging channels are fetched from the configured identity store.
- SMS and MOBILE notifications are sent to the mobile phone number.
- VOICE notifications are sent to the business phone number.
- No special notification is sent when the messaging channel preference is WORKLIST. Instead, log in to Oracle BPM Worklist to view tasks.
- EMAIL is the default messaging channel preference when a preferred channel has not been selected.

You can use the **Messaging Channels** tab to view, create, edit, and delete messaging channels.

29.9.3.1 Viewing Your Messaging Channels

You can display your existing messaging channels.

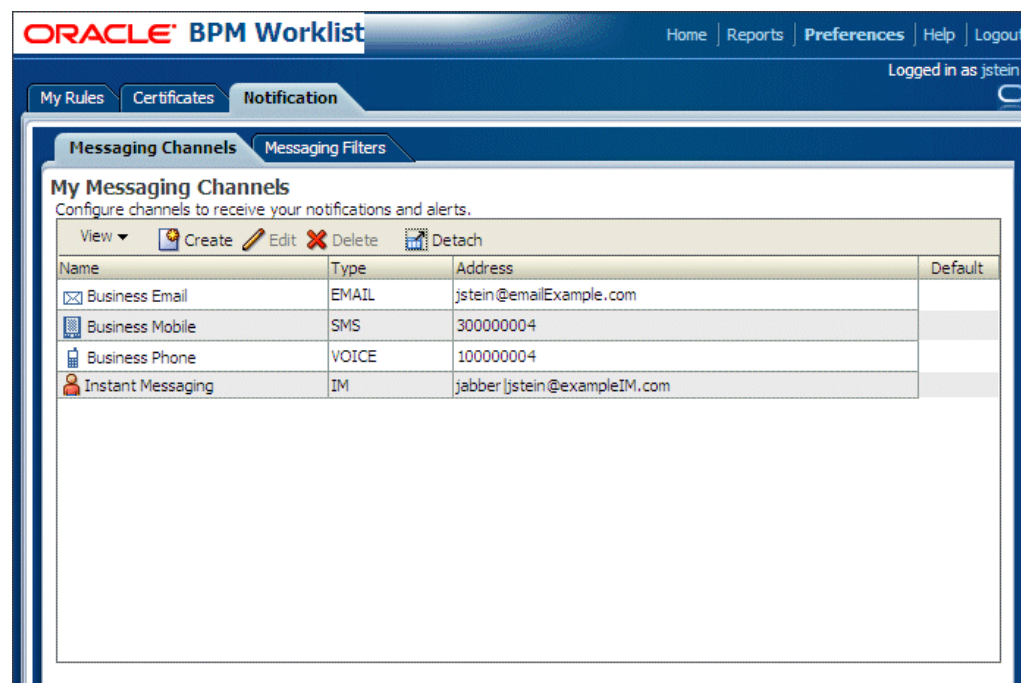
To view messaging channels:

1. Click the **Preferences** link.
2. Click the **Notification** tab.
3. Click the **Messaging Channels** tab.

The My Messaging Channels list appears (Figure 29–52) and displays the following information:

- **Name:** The name of the messaging channel.
- **Type:** The type of messaging channel, such as EMAIL or SMS.
- **Address:** The address for the channel, such as a phone number or email address.
- **Default:** Specifies whether this channel is the default messaging channel.

Figure 29–52 *Messaging Channels*



4. Click **View > Columns** and select the columns to display or hide.

You can also click **View > Reorder Columns** to display a dialog to reorder the displayed columns.

Messaging channel names and addresses are retrieved from the underlying identity store, such as Oracle Internet Directory.

29.9.3.2 Creating, Editing, and Deleting a Messaging Channel

Oracle BPM Worklist uses an underlying identity store, such as Oracle Internet Directory, to manage messaging channels and addresses. Therefore, you cannot directly create, modify, or delete messaging channels using Oracle BPM Worklist.

To perform these actions, contact the system administrator responsible for managing your organization's identity store.

29.9.4 Managing Messaging Filters

You can use the **Messaging Filters** tab to define filters that specify the types of notifications you want to receive along with the channels through which to receive these notifications. You can do this through a combination of comparison operators (such as *is equal to*, *is not equal to*), attributes that describe the notification type, content, or source, and notification actions, which send the notifications to the first available messaging channels, all messaging channels, or to no channels (effectively blocking the notification).

For example, you can create a messaging filter called *Messages from Lise*, that retrieves all messages addressed to you from your boss, Lise. Notifications that match all of the filter conditions might first be directed to your business mobile phone, for instance, and then to your business email if the first messaging channel is unavailable.

29.9.4.1 Viewing Messaging Filters

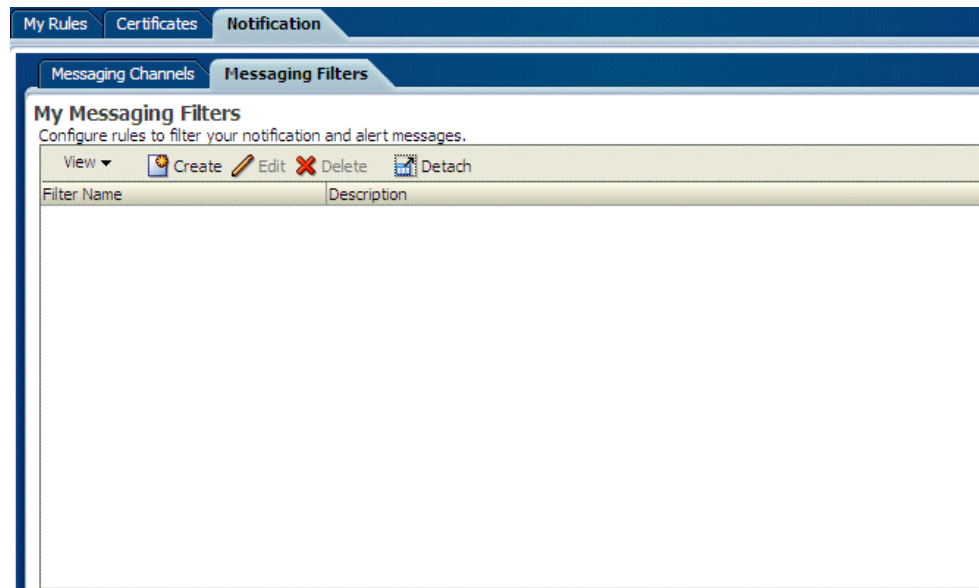
You can display your existing messaging filters.

To view your messaging filters:

1. Click the **Notification** tab.
2. Click the **Messaging Filters** tab.

The My Messaging Filters list appears ([Figure 29-53](#)) and displays the following information:

- **Filter Name:** The name of the messaging filter
- **Description:** An optional description of the messaging filter

Figure 29–53 Messaging Filters

3. Click **View > Columns** and select the columns to display or hide.

You can also click **View > Reorder Columns** to display a dialog to reorder the displayed columns.

29.9.4.2 Creating Messaging Filters

To create a messaging filter:

1. Click **Create**.

The Messaging Filters page appears, as shown in [Figure 29–54](#).

Figure 29–54 Adding a Messaging Filter

Oracle BPM Worklist

Home | Reports | Preferences | Help | Logout

Logged in as jstein

My Rules | Certificates | **Notification**

Messaging Channels | **Messaging Filters**

Your Reference System Time: Thursday, April 9, 2009 6:21:14 PM PDT

* Filter Name:

Description:

Condition

Matching: All of the following conditions

Add Filter Condition: Status isEqual *

Attribute	Operator	Value	Value2 (if required)	Delete

Action

Messaging Option: Send No Messages

Add Notification Channel: Instant Messaging

Channel	Address	Up	Down	Delete

2. Specify the following information:
 - **Filter Name:** The name of the messaging filter.
 - **Description:** An optional description for the messaging filter.
3. Define the filter conditions using the lists and fields in the **Condition** section, as follows:
 - a. Select whether notifications must meet all of the conditions or any of the conditions by selecting either the **All of the following conditions** or the **Any of the following conditions** options.
 - b. Select the attribute from the list.
 - c. Select the operator, such as **isEqual**, from the list.
 - d. Type the value of the condition in the text box.
 - e. Click **Add** to add the condition to the list.
 - f. Repeat these steps to add more filter conditions. To remove a filter condition, click **Delete**.
4. Select from the following messaging options in the **Action** section:
 - **Send No Messages:** Do not send a message to any channel.
 - **Send Messages to All Selected Channels:** Send a message to all specified channels in the address list.

- **Send to the First Available Channel:** Send a message serially to channels in the address list until one successful message is sent. This entails performing a send to the next channel when the current channel returns a failure status.
- 5. To set the delivery channel, select a channel from the **Add Notification Channel** list and click **Add**. To remove a channel, click **Delete**.
- 6. Use the up and down arrows to prioritize channels. If available, the top-most channel receives messages meeting the filter criteria if you select *Send to the First Available Channel*.
- 7. Click **OK**.

The messaging filter appears on the My Messaging Filters page. The My Messaging Filters page enables you to edit or delete the channel. Click **Cancel** to dismiss the dialog without creating the filter.

29.9.4.3 Editing a Messaging Filter

To edit a messaging filter:

1. Select the filter on the My Messaging Filters page.
2. Click **Edit**.
3. Click **OK** to update the messaging filter. Click **Cancel** to dismiss the dialog without modifying the filter.

29.9.4.4 Deleting a Messaging Filter

To delete a messaging filter:

1. Select the filter on the My Messaging Filters page.
2. Click **Delete**. A confirmation dialog appears.
3. Click **OK** to delete the messaging filter. Click **Cancel** to dismiss the dialog without deleting the filter.

29.10 Using Mapped Attributes (Flex Fields)

Human workflow mapped attributes (formerly referred to as flex fields) store and query use case-specific custom attributes. These custom attributes typically come from the task payload values. Storing custom attributes in mapped attributes provides the following benefits:

- They can be displayed as a column in the task listing
- They can filter tasks in custom views and advanced searches
- They can be used for a keyword-based search

For example the `Requester`, `PurchaseOrderID`, and `Amount` fields in a purchase order request payload of a task can be stored in the mapped attributes. An approver logging into Oracle BPM Worklist can see these fields as column values in the task list and decide which task to access. The user can define views that filter tasks based on the mapped attributes. For example, a user can create views for purchase order approvals based on different amount ranges. If the user must also retrieve tasks at some point related to a specific requester or a purchase order ID, they can specify this in the keyword field and perform a search to retrieve the relevant tasks.

For the mapped attributes to be populated, an administrator must create mapped attribute mappings, as follows:

1. Specify a label for the mapped attribute to be populated.
2. Map the payload attribute containing the data to the label.

These mappings are valid for a certain task type. Therefore, each task type can have different mapped attribute mappings. After the mapping is complete and any new task is initiated, the value of the payload is promoted to the mapped attribute. Tasks initiated before the mapping do not contain the value in the mapped attribute. Only top-level simple type attributes in the payload can be promoted to a mapped attribute. Complex attributes or simple types nested inside a complex attribute cannot be promoted. It is important to define the payload for a task in the Human Task Editor, keeping in mind which attributes from the payload may must promoted to a mapped attribute. All text and number mapped attributes are automatically included in the keyword-based search.

Essentially, the Human Task Editor is used only when defining the payload for a task. All other operations are performed at runtime.

Directory naming is not available concomitant with the flex file naming convention.

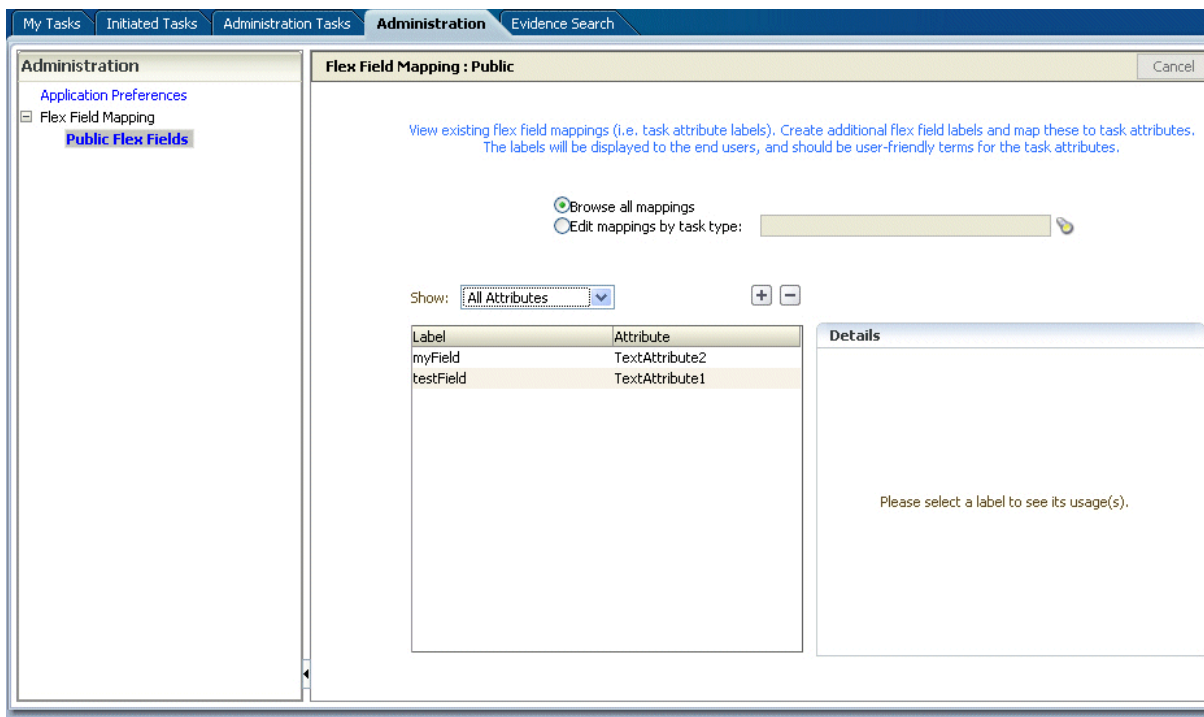
Note:

- Mapped attributes must be defined before instances of the business process are generated. Only instances generated after mapped attributes are created reflect the correct mapped attributes. Older instances of the business process do not reflect subsequent mapped attribute changes.
 - When you add a new locale, the mapped attribute labels are not automatically translated until you have flushed the cache. You may flush the cache either by restarting the server, or by changing a value in the workflow configuration settings—for example, by changing the `workflowCustomClasspathURL` property in the workflow configuration to some new value, then changing it back again.
-

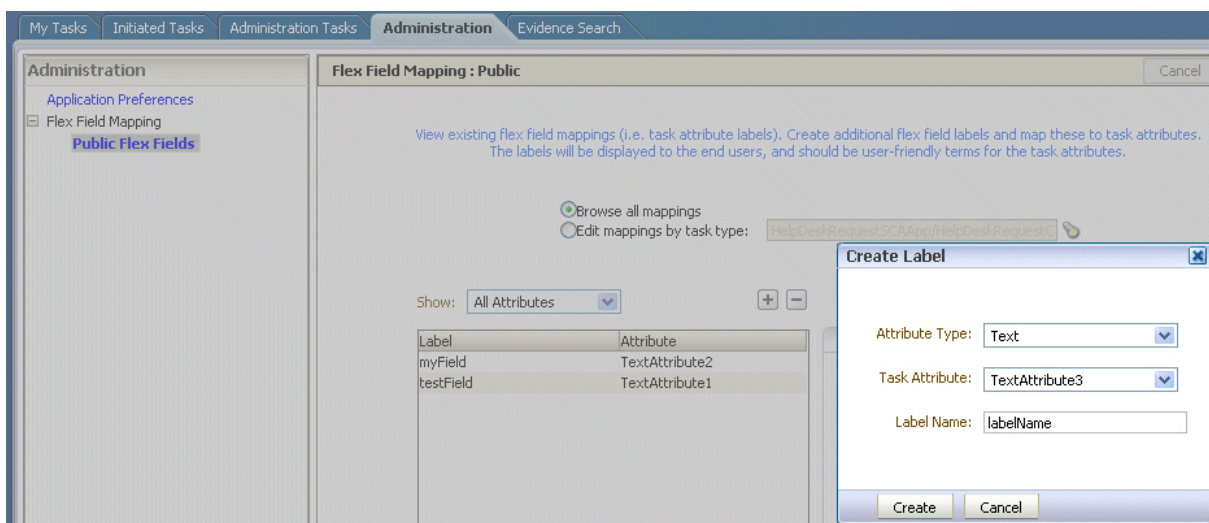
29.10.1 How To Map Attributes

An administrator, or users with special privileges, can use attribute mapping, shown in [Figure 29-55](#), to promote data from the payload to inline mapped attributes. By promoting data to mapped attributes, the data becomes searchable and can be displayed as columns on the task list page.

Administrators can map public mapped attributes. Users who have been granted the `workflow.mapping.publicFlexField` privilege can map public mapped attributes, and see a **Public Flex Fields** node on the **Administration** tab.

Figure 29–55 Mapped Attribute Mapping**To create labels:**

To create a mapped attribute mapping, an administrator first defines a semantic label, which provides a more meaningful display name for the mapped attribute. Click **Add** to use the Create Label dialog, as shown in Figure 29–56.

Figure 29–56 Creating a Label

As Figure 29–56 shows, **labelName** is mapped to the task attribute **TextAttribute3**. The payload attribute is also mapped to the label. In this example, the **Text** attribute type is associated with **labelName**. The result is that the value of the **Text** attribute is stored in the **TextAttribute3** column, and **labelName** is the column label displayed in the user's

task list. Labels can be reused for different task types. You can delete a label only if it is not used in any mappings.

A mapped payload attribute can also be displayed as a column in a custom view, and used as a filter condition in both custom views and workflow rules. The display name of the payload attribute is the attribute label that is selected when doing the mapping.

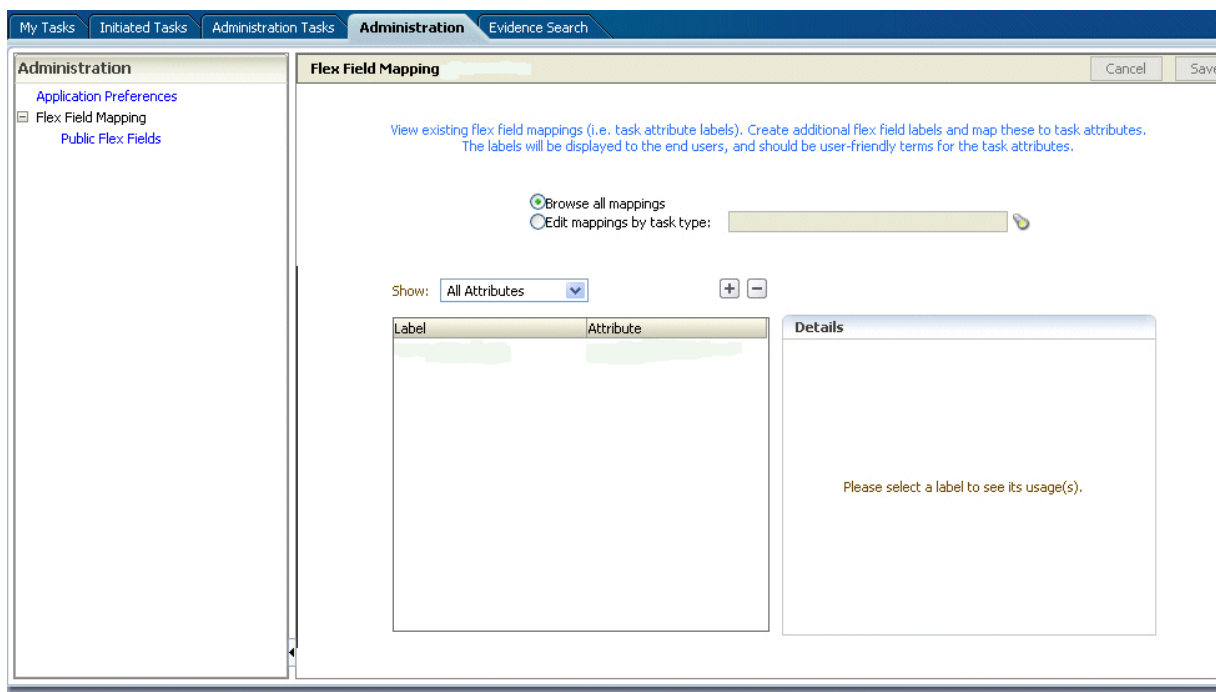
Note the following restrictions:

- Only simple type payload attributes can be mapped.
- A mapped attribute (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:

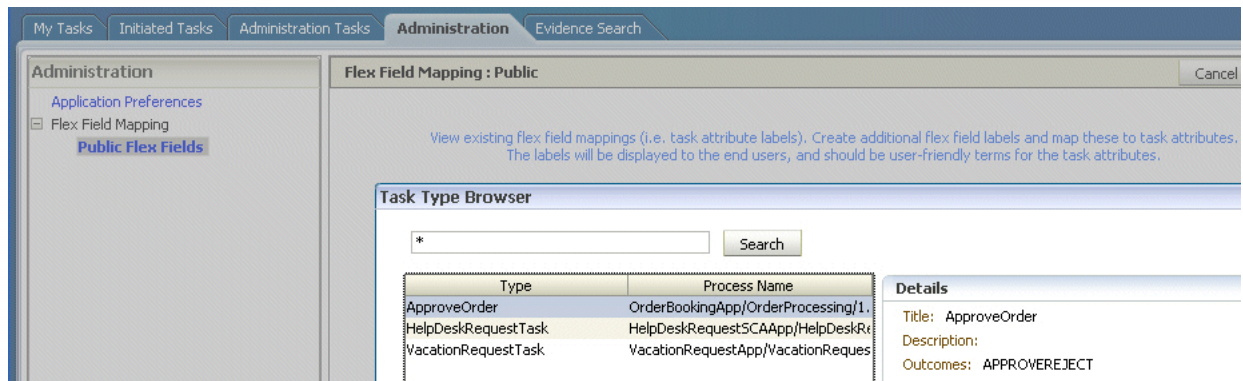
1. Click **Browse all mappings**.
2. Select a row in the label table to display all the payload attributes mapped to a particular label.

Figure 29–57 Browsing Mappings

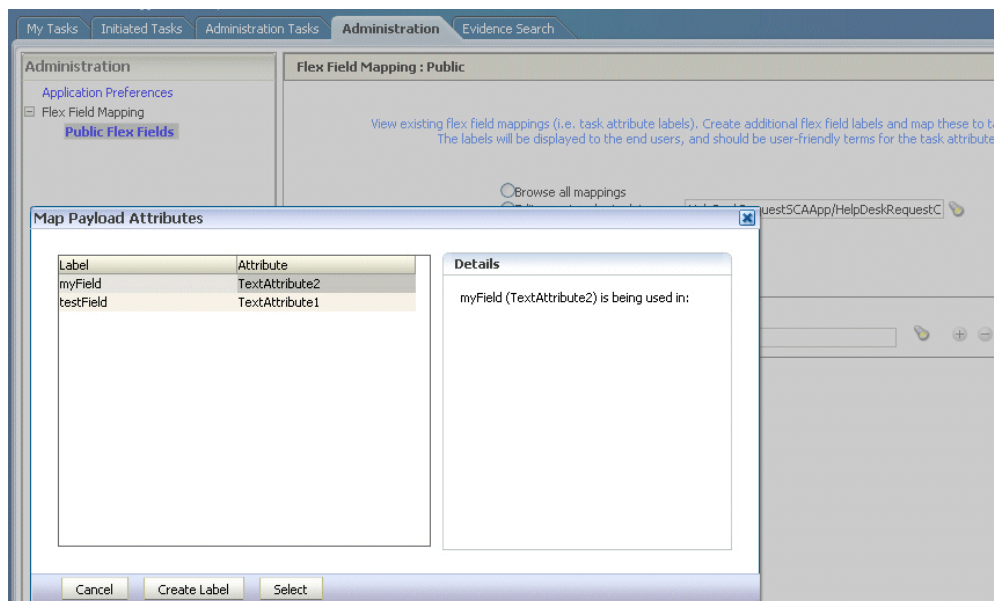


To edit mappings by task type:

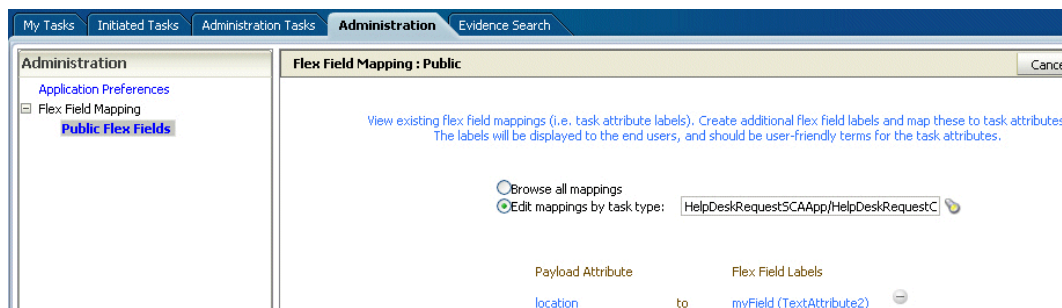
1. Click **Edit mappings by task type**, optionally provide a task type, and click **Search**.
2. Select a task type and click **OK**.

Figure 29–58 Selecting a Task Type

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 29–59](#).

Figure 29–59 Selecting a Label

4. Select a mapping label and click **Select**.
[Figure 29–60](#) shows a completed mapping.

Figure 29–60 Mapped Attribute Mapping Created

See [Section 31.1.9.1, "Internationalization of Attribute Labels"](#) for more information.

29.10.2 Custom Mapped Attributes

The following mapped attributes are included in the `WorkflowTask.xsd` file and are available for your use without restrictions.

Table 29–11 Custom Mapped Attributes

Attribute	Datatype
<code>customerAttributeString1</code>	String
<code>customerAttributeString2</code>	String
<code>customerAttributeNumber1</code>	Double
<code>customerAttributeNumber2</code>	Double
<code>customerAttributeDate1</code>	Date
<code>customerAttributeDate2</code>	Date

Use the following Java Architecture for XML Binding (JAXB) methods to set and get these attributes:

```
task.getCustomerAttributes().getCustomerAttributeString1()
task.getCustomerAttributes().setCustomerAttributeString1("String")
task.getCustomerAttributes().getCustomerAttributeNumber1()
task.getCustomerAttributes().setCustomerAttributeNumber2(9)
task.getCustomerAttributes().setCustomerAttributeDate1()
task.getCustomerAttributes().setCustomerAttributeDate2()
```

These fields are persisted in the database as `customerAttributeString1`, `customerAttributeString2`, `customerAttributeNumber1`, `customerAttributeNumber2`, `customerAttributeDate1`, `customerAttributeDate2`.

29.11 Creating Worklist Reports

[Table 29–12](#) lists the worklist reports available for task analysis.

Table 29–12 Worklist Report Types

Report Name	Description	Input Parameters
Unattended Tasks	Provides an analysis of tasks assigned to users' groups or reportees' groups that have not yet been acquired (the "unattended" tasks).	- Assignee—This option (required) selects tasks assigned to the user's group (My Group), tasks assigned to the reportee's groups (Reportees), tasks where the user is a creator (Creator), or tasks where the user is an owner (Owner). - Creation Date—An optional date range - Expiration Date—An optional date range - Task State—The state (optional) can be Any, Assigned, Expired, or Information Requested. - Priority—The priority (optional) can be Any, Highest, High, Normal, Low, or Lowest.
Tasks Priority	Provides an analysis of the number of tasks assigned to a user, reportees, or their groups, broken down by priority.	- Assignee—Depending on the assignee that you select, this required option includes tasks assigned to the logged-in user (My), tasks assigned to the user and groups that the user belongs to (My & Group), or tasks assigned to groups to which the user's reportees belong (Reportees). - Creation Date—An optional date range - Ended Date—An optional date range for the end dates of the tasks to be included in the report - Priority—The priority (optional) can be Any, Highest, High, Normal, Low, or Lowest.
Tasks Cycle Time	Provides an analysis of the time taken to complete tasks from assignment to completion based on users' groups or reportees' groups.	- Assignee—Depending on the assignee that you select, this required option includes your tasks (My) or tasks assigned to groups to which your reportees belong (Reportees). - Creation Date—An optional date range - Ended Date—An optional date range for the end dates of the tasks to be included in the report - Priority—The priority (optional) can be Any, Highest, High, Normal, Low, or Lowest.
Tasks Productivity	Provides an analysis of assigned tasks and completed tasks in a given time period for a user, reportees, or their groups.	- Assignee—Depending on the assignee that the user selects, this required option includes the user's tasks (My & Group) or tasks assigned to groups to which the user's reportees belong (Reportees). - Creation Date (range)—An optional creation date range. The default is one week. - Task Type—Use the Search (flashlight) icon to select from a list of task titles. All versions of a task are listed on the Select Workflow Task Type page (optional).
Tasks Time Distribution	Provides the time an assignee takes to perform a task.	- Assignee—Depending on the assignee that the user selects, this required option includes the user's tasks (My & Group) or tasks assigned to groups to which the user's reportees belong (Reportees). - From...to (date range)—An optional creation date range. The default is one week. - Task Type—Use the Search (flashlight) icon to select from a list of task titles. All versions of a task are listed on the Select Workflow Task Type page (optional).

29.11.1 How To Create Reports

Reports are available from the **Reports** link. Report results cannot be saved.

To create a report:

1. Click the **Reports** link.
2. Click the type of report you want to create.

[Figure 29–61](#) shows the report types available.

Figure 29–61 Oracle BPM Worklist Reports

3. Provide inputs to define the search parameters of the report.

Figure 29–62 shows an example of the Unattended Tasks Report input page. The other reports are similar. See Table 29–12 for information about input parameters for all the report types.

Figure 29–62 Unattended Tasks Report—Input Page for Task Analysis

Unattended Tasks Report
Provides an analysis of tasks assigned to users' groups or reportees' groups that need attention because they have not yet been acquired

Assignee:

Creation Date: to

Expiration Date: to

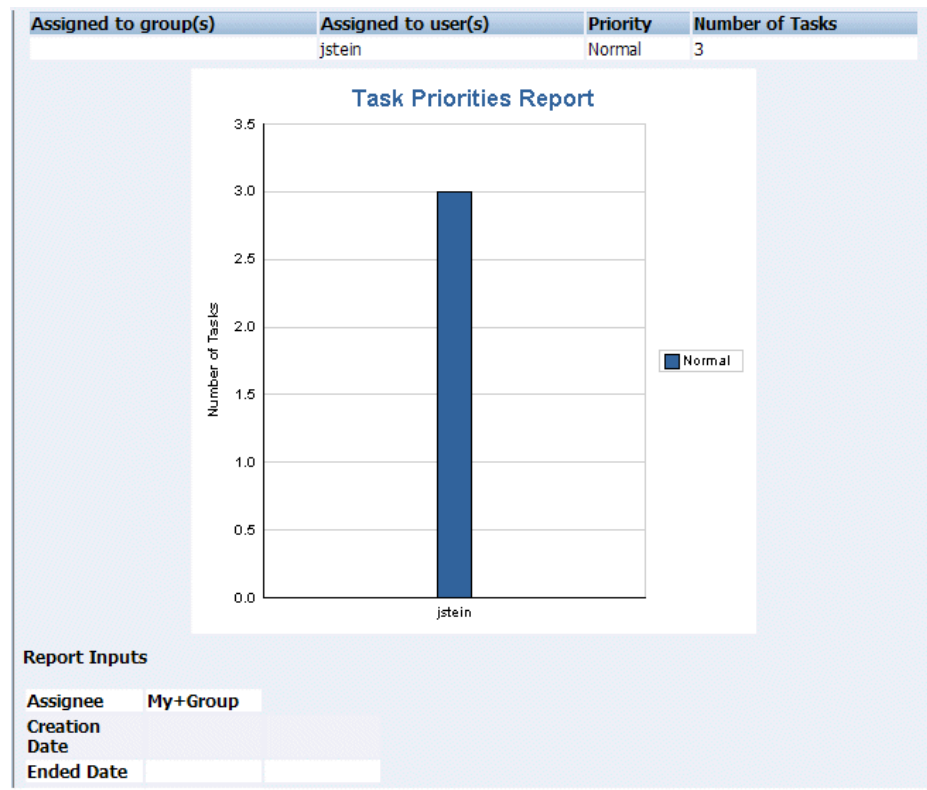
Task State:

Priority:

4. Click **Run**.

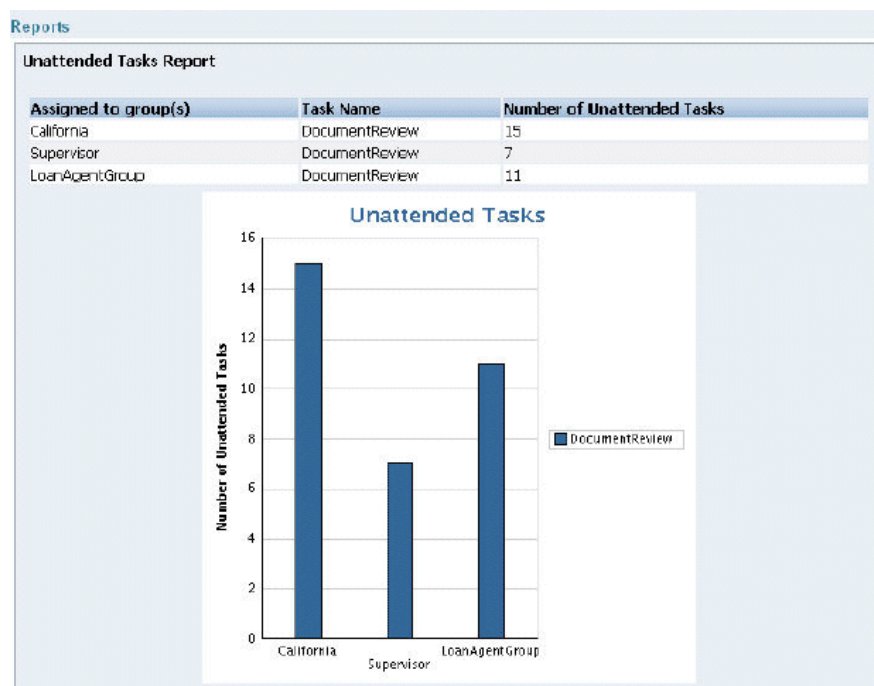
29.11.2 What Happens When You Create Reports

As shown in Figure 29–63, report results (for all report types) are displayed in both a table format and a bar chart format. The input parameters used to run the report are displayed under **Report Inputs**, in the lower-left corner (may require scrolling to view).

Figure 29–63 Report Display—Table Format, Bar Chart Format, and Report Inputs

29.11.2.1 Unattended Tasks Report

Figure 29–64 shows an example of an Unattended Tasks report.

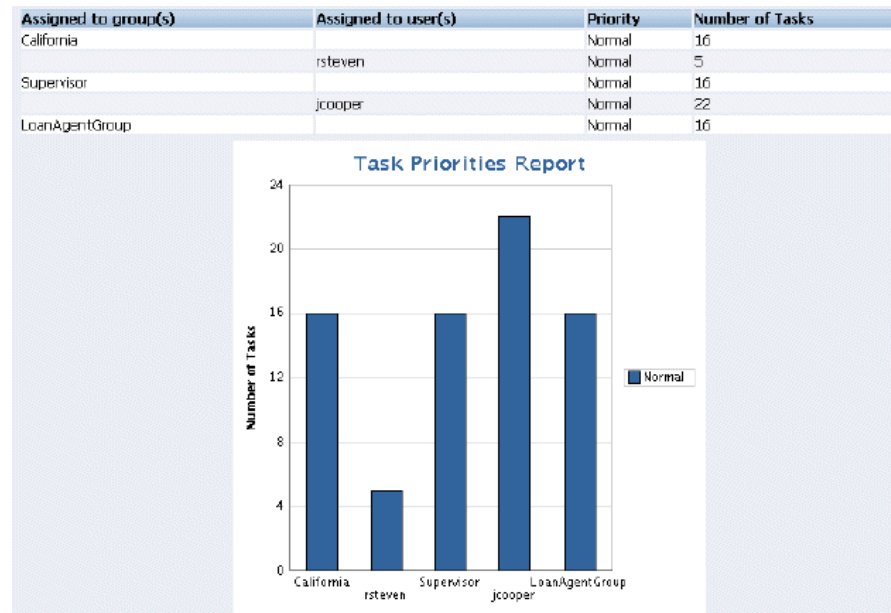
Figure 29–64 Unattended Tasks Report

The report shows that the California group has 15 unattended tasks, the Supervisor group has 7 unattended tasks, and the LoanAgentGroup has 11 unattended tasks. The unattended (unclaimed) tasks in this report are all DocumentReview tasks. If multiple types of unattended task exists when a report is run, all task types are included in the report, and the various task types are differentiated by color.

29.11.2.2 Tasks Priority Report

Figure 29–65 shows an example of a Tasks Priority report.

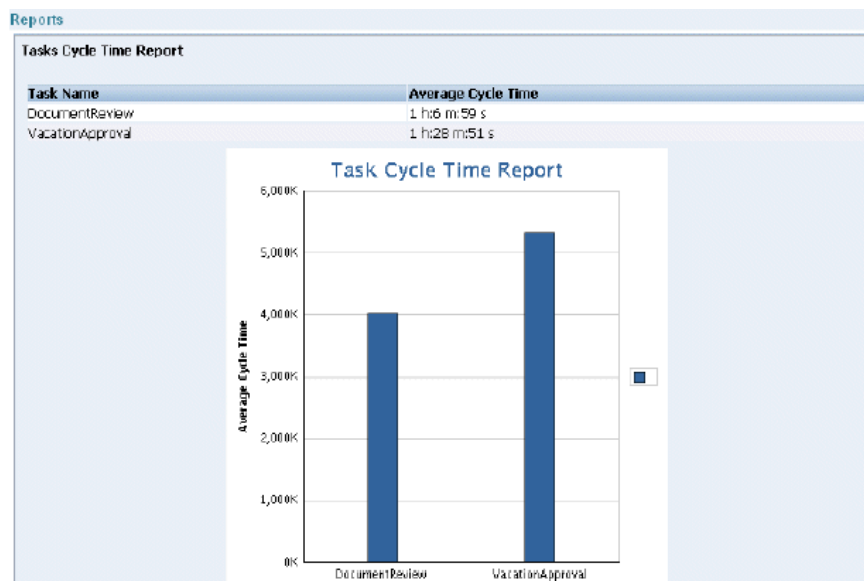
Figure 29–65 Tasks Priority Report



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.

29.11.2.3 Tasks Cycle Time Report

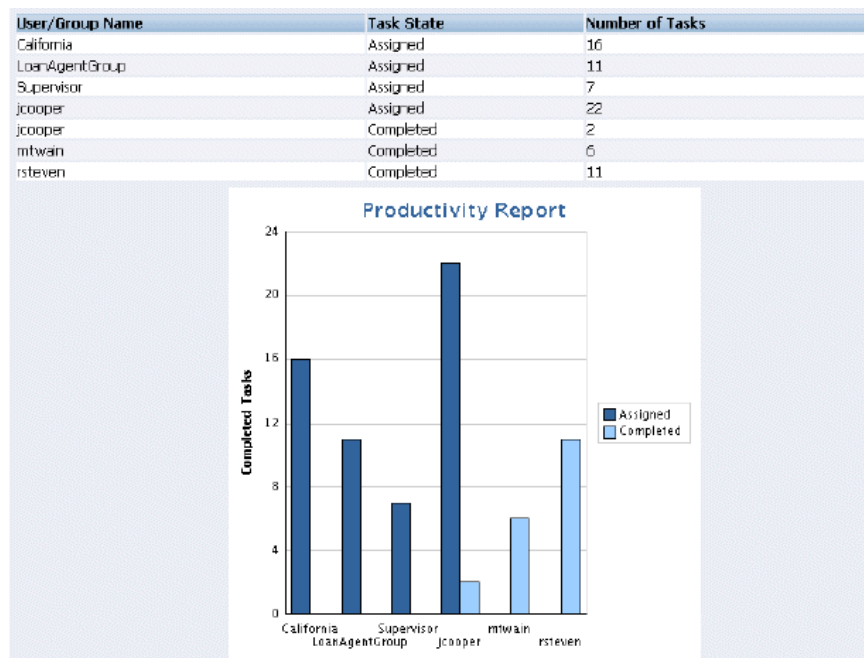
Figure 29–66 shows an example of a Tasks Cycle Time Report.

Figure 29–66 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.

29.11.2.4 Tasks Productivity Report

Figure 29–67 shows an example of a Tasks Productivity Report.

Figure 29–67 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.

Note: The **Me and Group** and **Reportees** options have been removed from the Productivity Report.

29.12 Accessing Oracle BPM Worklist in Local Languages and Time Zones

A user's preferred worklist language is configured from either the identity store or the browser.

A user's preferred time zone is configured from the identity store.

If no preference information is available, then the user's preferred language and time zone are determined by the system defaults. System defaults are based on the server settings for language and time zone.

If the custom resource bundle class in the browser locale is *not* available and the custom resource bundle class in the default server locale *is* available, then the language is derived from the latter, namely, custom resource bundle class in the default server locale.

If the custom resource bundle class in default server locale is also not available, then the language is derived from the custom base class.

If no user language preferences are set, or if they are set to a language not supported by Oracle BPM Worklist, then the Worklist Application defaults to English.

For more information, see the following sections for instructions on how to select **Browser** or **Identity Provider** in the worklist interface:

- [Section 29.8.4, "How to Specify the Language Locale Information"](#) for how to select **Browser** or **Identity Provider** from the Application Preferences page
- [Section 29.3, "Customizing the Task List Page"](#) and [Figure 29–14, "Customizing Fields in a Worklist View"](#)

29.12.1 Strings in Oracle BPM Worklist

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 29.8.3, "How to Specify the Resource Bundle"](#)) or by providing an updated version of the default bundle class. See the Workflow Customizations sample for details.

For task attribute names, mapped 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 31, "Introduction to Human Workflow Services"](#) for information on adding entries to this file for dynamic assignment functions and attribute labels.

For custom actions and task titles, the display names come from the message bundle specified in the task configuration file. If no message bundle is specified, then the values specified at design time are used. See [Chapter 31, "Introduction to Human Workflow Services"](#) for information on how to specify message bundles so that custom actions and task titles are displayed in the preferred language.

29.12.2 How to Change the Preferred Language if the Identity Store is LDAP-Based

If an LDAP-based provider such as Oracle Internet Directory is used, then language settings are changed in the Oracle Internet Directory community. Connect to the embedded LDAP server, where you can change language settings in the Oracle Internet Directory community.

1. Start an LDAP browser (for example, openLdap browser, ldapbrowser, jXplorer, and so on). See the documentation for your browser for instructions.
2. Connect to the LDAP server by providing the hostname, port number on which the server is running, and the administration user credentials with which to log in.

- For Embedded LDAP:

- a. The default managed server port number is 7001.
- b. The administration credential username is cn=admin.
- c. The administration password credential is accessible from the Oracle WebLogic Server Administration Console by selecting **Security > Embedded LDAP** for your domain.

For instructions on changing the default password credential, see Chapter 9, "Managing the Embedded LDAP Server" of *Oracle Fusion Middleware Securing Oracle WebLogic Server*.

- For OIDm:

- a. The default port number is 3060.
- b. The administration username is cn=orcladmin.
- c. The administration password is the password for the LDAP server.

3. To change a user's preferred language, navigate to the user entry, and add/set the preferredLanguage attribute. See [Table 29–13, "Languages Supported in Oracle BPM Worklist"](#) for a list of supported languages. To change the time zone setting, add/set the orclTimeZone attribute. The format of the time zone string is Continent/Region. You can find the time zone values in the \$JAVA_HOME/jre/lib/zi directory. The directories specify the continent names, for example, Africa, Asia, America, and so on, while the files within the directories specify the regions. Note that some regions include subregions, for example America/Indiana/Indianapolis.

When a user logs in, the worklist pages are rendered in the user's preferred language and time zone.

29.12.3 How to Change the Language in Which Tasks Are Displayed

For better performance, only the English language is listed for the LocaleList property in the System MBean Browser in Oracle Enterprise Manager Fusion Middleware Control. If you want to display the task title, category, and subcategory in other languages or add other languages, you must change the required language locale in the System MBean Browser.

Note: You should add all languages at the very beginning. If you add another language later, then any tasks previously written in other languages no longer appear in the worklist. For example, if the previously specified language was English, and you later added French, then any tasks written before you added French no longer appear in the worklist.

To add or change a language:

1. In Oracle Enterprise Manager Fusion Middleware Control, right-click `soa-infra` in the navigator, select Administration, then select System MBean Browser.
2. Expand the following in sequence: Application Defined MBeans; then `oracle.as.soainfra.config`; then Server: `server_name`; then WorkflowConfig.
3. Click human-workflow.

To change the language:

- a. In the Name column, click `LocaleList`.
- b. In the Value field, click the value.
- c. In the Name column, click `Language`.
- d. In the Value field, change `en` to the language value that you want to use.
- e. Click **Apply**.

To add additional languages:

- a. Click the Operations tab.
- b. In the Name column, click `createLocale`.
- c. In the Value field, enter a value. For better performance, ensure that you include only the languages that you need for task title, category, and subcategory.
- d. Click **Invoke**.

29.12.4 How To Change the Language Preferences from a JAZN XML File

In the JAZN XML file, change the portion in bold to set the user's preferred language.

```
<preferredLanguage>en-US</preferredLanguage>
```

Oracle BPM Worklist supports the languages shown in [Table 29–13](#).

Table 29–13 Languages Supported in Oracle BPM Worklist

Language	Format
English	(en)
English (United States)	(en-US)
German	(de)
Spanish (International)	(es)
French	(fr)
Italian	(it)
Japanese	(ja)

Table 29–13 (Cont.) Languages Supported in Oracle BPM Worklist

Language	Format
Korean	(ko)
Portuguese (Brazil)	(pt-BR)
Chinese (Simplified)	(zh-CN)
Chinese (Traditional)	(zh-TW)

29.12.5 What You May Need to Know About Runtime Languages Not Displaying in the Worklist

Oracle BPM Worklist supports nine administration languages. However, the user's notification preference interface, as a standalone application, supports 21 runtime languages. If a user's preferred language is set to a language that is not supported by the worklist, but which is supported by the user's notification preference interface, then the worklist displays the language set by the server (or English if the server language is also not supported by the worklist), while the embedded user's notification preference interface displays in the user's preferred language. In this case, two languages are seen when you navigate to the Preferences settings in the **Notification** tab in the worklist.

For example, assume that the language of the SOA server is French and that someone tries to access the worklist in a browser with the language set to Arabic. The worklist interface displays the server language, French, while the embedded user's notification preference interface displays in Arabic when navigating to the **Preferences > Notification** tab.

29.12.6 What You May Need to Know About Inconsistent Display Languages in Worklist and Embedded User's Notification Preference Interface

Oracle BPM Worklist can be configured to set the language from the browser or from the identity store. There are two levels to this setting: the application level and the user level. If the user preference is set, it takes precedence in determining the worklist display language. However, the embedded user's notification preference interface always respects the application preference. Therefore, if the user's preference indicates that the language from the browser is to be used, while the application preference is set to use the language from the identity store, or vice versa, you may see different display languages in the worklist and in the user's notification preference interface.

29.12.7 How To Change the Time Zone Used in the Worklist

The following is based on extracting a user's time zone from a JAZN XML file.

To change the time zone:

Change the string in bold to set the user's preferred time zone.

```
<timeZone>America/Los_Angeles</timeZone>
```

The format of the time zone string is *Continent/Region*. You can find the time zone values in the `$JAVA_HOME/jre/lib/zi` directory. The directories specify the continent names, for example Africa, Asia, America, and so on, while the files within the directories specify the regions. Note that some regions include sub-regions, for example `America/Indiana/Indianapolis`.

29.13 Creating Reusable Worklist Regions

Some features available in worklist are exposed as standalone reusable components that can be embedded in any application. Moreover, these standalone task flows provide many customizations through parameters that enable user to build and customize a worklist application to meet requirements. All of the task flows are bundled in an ADF library that can be included in the embedding application.

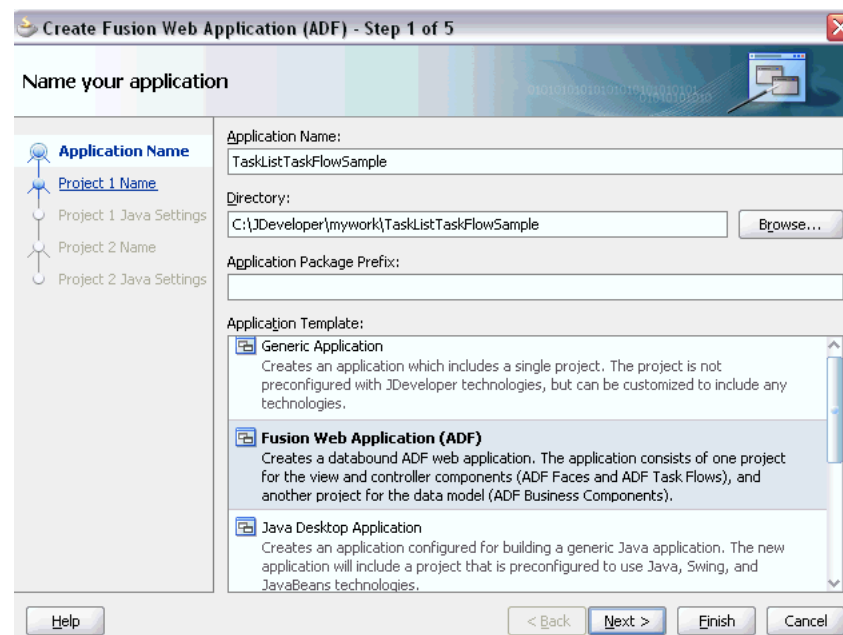
29.13.1 How to Create an Application With an Embedded Reusable Worklist Region

The usage of each reusable worklist region is the same with a few exceptions. The following procedure provides the detailed steps to create an application and embed the Task List task flow in the application. Where applicable, notes on how to use other types of reusable worklist regions are provided.

To create an application with an embedded reusable worklist region:

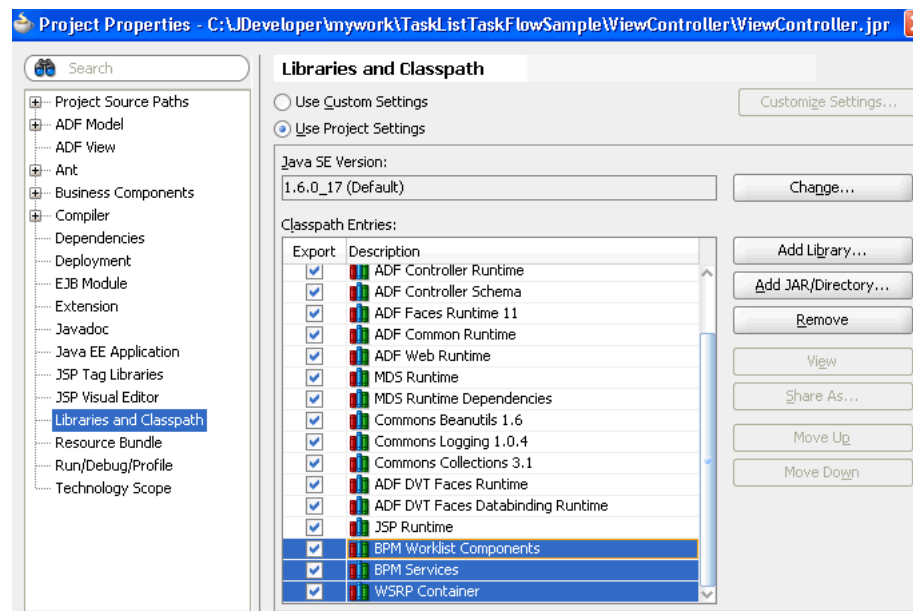
1. Create new Fusion Web Application in Oracle JDeveloper. In this example, the name of the application is TaskListTaskFlowSample. [Figure 29–68](#) provides details.

Figure 29–68 Creation of Application with an Embedded Reusable Worklist Region



2. Open the View Controller Project Properties, **Libraries and Classpath** section, and click **Add Library** to add the following libraries in the class path:
 - **BPM Worklist Components**
Add this library to add the task flow JAR `adflibTaskListTaskFlow.jar` and `adflibWorklistComponents.jar`, which are required in the project's class path.
 - **BPM Services**
 - **WSRP Container**

[Figure 29–69](#) provides details.

Figure 29–69 Libraries and Classpath Section

3. If your application runs on non-SOA server, you must perform two additional steps.

- a. Install the `oracle.soa.workflow` shared library.

Note that if your server has `oracle.soa.workflow.wc` already installed you do not need to install `oracle.soa.workflow`.

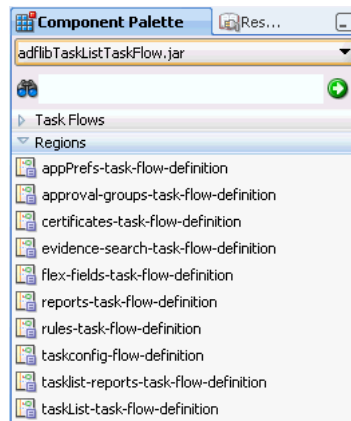
- b. Configure a foreign JNDI on the server.

Note that if you run the Task List task flow in federated mode, you do not need to do this step. See the section "[federatedMode](#)" on page 29-80 for information about how to use the task flow in federated mode.

4. Select the View Controller project and choose **File > New > Current Project Technologies > Web Tier > JSF Page** to create a jsp file (for example, `testSample.jsp`).

Be sure to select **Create as XML document (*.jspx)** in the Create JSF Page dialog.

5. Choose `adflibTaskListTaskFlow.jar` from the component palette. It contains the list of all the Task Flows and Regions. [Figure 29–70](#) provides details.

Figure 29–70 Component Palette

6. Drag and drop one of the task flow Regions to the jsp page, and select **Region** in the **Create** menu (for example, **taskList-task-flow-definition** for Task List Task Flow).

See the following sections for details about the task flow definitions:

- [Section 29.13.4, "What You May Need to Know About Task List Task Flow"](#)
 - [Section 29.13.5, "What You May Need to Know About Certificates Task Flow"](#)
 - [Section 29.13.6, "What You May Need to Know About the Reports Task Flow"](#)
 - [Section 29.13.7, "What You May Need to Know About Application Preferences Task Flow"](#)
 - [Section 29.13.8, "What You May Need to Know About Mapped Attributes Task Flow"](#)
 - [Section 29.13.9, "What You May Need to Know About Rules Task Flow"](#)
7. If you chose **flex-fields-task-flow-definition**, **rules-task-flow-definition**, **tasklist-reports-task-flow-definition**, or **taskList-task-flow-definition**, pass the task flow parameters in the Edit Task Flow Binding dialog that appears.
 8. A new entry is added to the `pagenamePagedef.xml` file.

For example, adding the **taskList-task-flow-definition** results in the following new entry:

```
<taskFlow id="taskListtaskflowdefinition1"

    taskFlowId="/WEB-INF/taskList-task-flow-definition.xml#taskList-task-
flow-definition"
    xmlns="http://xmlns.oracle.com/adf/controller/binding">
  <parameters>
    <parameter id="federatedMode" value="true"
      xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
    <parameter id="showServerColumn" value="true"
      xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  </parameters>
</taskFlow>
```

9. Add the shared libraries in the `weblogic-application.xml` file. If you have `oracle.soa.workflow.wc` installed on your server, add that library.

```
<library-ref>
  <library-name>oracle.soa.workflow</library-name>
```

</library-ref>

10. Before deploying the application, see [Section 29.13.2, "How to Set Up the Deployment Profile."](#)

29.13.2 How to Set Up the Deployment Profile

Before deploying the application, you must edit the deployment profile.

To edit the deployment profile

1. Select the View Controller project and choose **File > New > General > Deployment Profiles**, select **WAR File**, and click **OK**.
2. Select **WEB-INF/lib > Filters**, and check `adflibTaskListTaskFlow.jar`, `adflibWorklistComponents.jar` and `wsrp-container.jar`.

29.13.3 How to Prepare Federated Mode Task Flows For Deployment

If you are using the task flow in federated mode, you must pass the list of federated servers to the task flow. See ["federatedMode"](#) on page 29-80 for details.

If the task flow is used in the federated mode, then enable global trust between the federated servers. This is done so that the already authenticated user token is passed to all the federated servers passed.

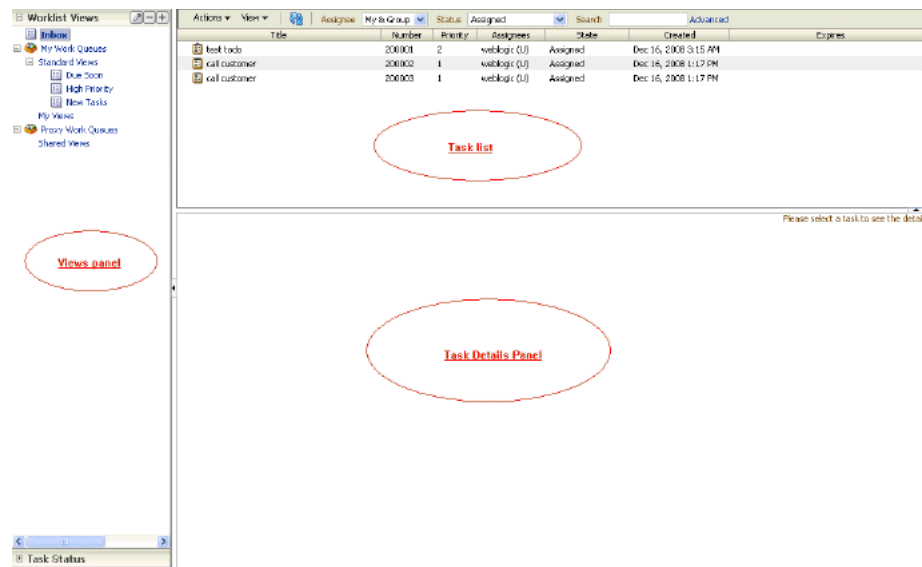
Do the below steps for all the federated servers and restart all the servers. It is very important that you restart all the servers.

To restart the servers:

1. Login to the Oracle Weblogic Server console.
2. Select the domain name **soainfra** under **Domain Structures**. The domain name may be different if a SOA server is not used.
3. Select the **Security** tab.
4. Select the **Advanced** link (near the bottom **Save** button).
5. Enter a password in the **Credential** field. (The same password must be given for all the federated servers).
6. Click **Save**.
7. Restart the server.

29.13.4 What You May Need to Know About Task List Task Flow

The Task List task flow takes in the parameters to control the display behavior of the embedded region. [Figure 29-71](#) provides details.

Figure 29–71 Task List

Some of the parameters are listed below. For the full list of parameters, see [Section 33.4, "Passing Worklist Portlet Parameters."](#)

- `federatedMode`
- `federatedServers`
- `showServerColumn`
- `wfCtxID`

federatedMode

If this is passed as true, the task list is shown in the federated mode. To run the task flow in federated mode, the list of federated servers must be passed to the task flow. You can pass the federated servers list to the task flow in one of the following two ways.

- Provide the client configuration file `wf_client_config.xml` in the class path (APP-INF\classes\wf_client_config.xml at the EAR level, or the WEB-INF\classes of the web application). The client configuration file contains all federated server details. See more information about this parameter in detail in [Section 33.4, "Passing Worklist Portlet Parameters."](#)
- Construct a JAXB object, which contains the federated servers list. This JAXB object can be passed to the task flow through the `federatedServers` parameter. See ["federatedServers"](#) on page 29-80 for information about constructing the JAXB object.

If both the client configuration file (`wf_client_config.xml`) and the JAXB object were provided to the task flow, the JAXB object takes the precedence.

federatedServers

This parameter is a JAXB object that contains the list of servers if the task flow is run in federated mode. This parameter takes precedence over the client configuration file (`wf_client_config.xml`) if it were also provided. See the code example in [Example 29–1](#) for details about constructing the JAXB object (`WorkflowServicesClientConfigurationType`).

Make sure that you set one of the servers as default, as shown in [Example 29–1](#). Only one server is required to be designated as the default. Also, verify that the server you designate as the default is excluded from the federated servers list. The relevant code for doing this is in bold in the example.

The default server is used when you have many servers defined in `wf_client_config.xml` or in the JAXB object, but the workflow client is desired for a single server. There are a few legacy APIs that do not take a server name as a parameter. To support such legacy APIs, you must define a single server as the default server, otherwise any legacy APIs that do not take a server name do not work.

Example 29–1 federatedServers

```
import oracle.bpel.services.workflow.client.config.IdentityPropagationType;
import oracle.bpel.services.workflow.client.config.PolicyReferenceType;
import oracle.bpel.services.workflow.client.config.PolicyReferencesType;
import oracle.bpel.services.workflow.client.config.RemoteClientType;
import oracle.bpel.services.workflow.client.config.ServerType;
import oracle.bpel.services.workflow.client.config.SoapClientType;
import
oracle.bpel.services.workflow.client.config.WorkflowServicesClientConfigurationType;

WorkflowServicesClientConfigurationType wscct =
    new WorkflowServicesClientConfigurationType();

List<ServerType> servers = wscct.getServer();

/**** Setting default server in the list ****/

ServerType defalutServer = new ServerType();
servers.add(defalutServer);

defalutServer.setDefault(true);
defalutServer.setExcludeFromFederatedList(true);
defalutServer.setName("default");

RemoteClientType rct = new RemoteClientType();
rct.setServerURL("t3://myhost.us.oracle.com:7001");
rct.setInitialContextFactory("weblogic.jndi.WLInitialContextFactory");
rct.setParticipateInClientTransaction(false);
defalutServer.setRemoteClient(rct);

SoapClientType sct = new SoapClientType();
PolicyReferencesType prts = new PolicyReferencesType();

PolicyReferenceType prt = new PolicyReferenceType();
prt.setEnabled(true);
prt.setCategory("security");
prt.setUri("oracle/wss10_saml_token_client_policy");
prts.getPolicyReference().add(prt);

IdentityPropagationType ipt = new IdentityPropagationType();
ipt.setMode("dynamic");
ipt.setType("saml");
ipt.setPolicyReferences(prts);

sct.setRootEndPointURL("http://myhost.us.oracle.com:7001");
sct.setIdentityPropagation(ipt);
```

```
defalutServer.setSoapClient(sct);

/***** Setting Federated Server 1 to the list *****/

ServerType server1 = new ServerType();
servers.add(server1);
server1.setName("Human Resource");

RemoteClientType rct1 = new RemoteClientType();
rct1.setServerURL("t3://stadl28.us.oracle.com:7001");
rct1.setInitialContextFactory("weblogic.jndi.WLInitialContextFactory");
rct1.setParticipateInClientTransaction(false);
server1.setRemoteClient(rct1);

SoapClientType sct1 = new SoapClientType();
PolicyReferencesType prts1 = new PolicyReferencesType();

PolicyReferenceType prt1 = new PolicyReferenceType();
prt1.setEnabled(true);
prt1.setCategory("security");
prt1.setUri("oracle/wss10_saml_token_client_policy");
prts1.getPolicyReference().add(prt1);
IdentityPropagationType ipt1 = new IdentityPropagationType();
ipt1.setMode("dynamic");
ipt1.setType("saml");
ipt1.setPolicyReferences(prts1);

sct1.setRootEndPointURL("http://stadl28.us.oracle.com:7001");
sct1.setIdentityPropagation(ipt1);

server1.setSoapClient(sct1);

/***** Setting Federated Server 2 to the list *****/

ServerType server2 = new ServerType();
servers.add(server2);
server2.setName("Financials");

RemoteClientType rct2 = new RemoteClientType();
rct2.setServerURL("t3://myhost.us.oracle.com:7001");
rct2.setInitialContextFactory("weblogic.jndi.WLInitialContextFactory");
rct2.setParticipateInClientTransaction(false);
server2.setRemoteClient(rct2);

SoapClientType sct2 = new SoapClientType();
PolicyReferencesType prts2 = new PolicyReferencesType();

PolicyReferenceType prt2 = new PolicyReferenceType();
prt2.setEnabled(true);
prt2.setCategory("security");
prt2.setUri("oracle/wss10_saml_token_client_policy");
prts2.getPolicyReference().add(prt2);

IdentityPropagationType ipt2 = new IdentityPropagationType();
ipt2.setMode("dynamic");
ipt2.setType("saml");
ipt2.setPolicyReferences(prts2);

sct2.setRootEndPointURL("http://myhost.us.oracle.com:7001");
sct2.setIdentityPropagation(ipt2);
```

```
server2.setSoapClient(sct2);
```

showServerColumn

If the task flow is run in federated mode, the server column in the task list is not shown by default. The server column is shown if this parameter is passed as `true`, otherwise it is not.

wfCtxID

This is a workflow context token string. It is used to create workflow context inside the task flow. If the application is SSO-enabled, or it is secured using ADF security, this parameter is not required, otherwise this is a required parameter. You can get the workflow context ID as shown in [Example 29–2](#).

Example 29–2 wfCtxID

```
IWorkflowContext wfCtx = wfSvcClient.getTaskQueryService().authenticate(username,
                                                                    password,
                                                                    realm,
                                                                    null);

wfCtxID = wfCtx.getToken();
```

29.13.5 What You May Need to Know About Certificates Task Flow

The user can upload the certificate to use to sign a decision, as shown in the following graphic. When signing a task outcome using your certificate, you must upload the entire chain of certificates through Oracle BPM Worklist as a .P7B (PKCS7 format) file, not only the one certificate issued to you by the certificate issuer.

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.

Certificates task flow does not have any parameters. [Figure 29–72](#) provides details.

Figure 29–72 Digital Certificate

Upload Certificate

☒ Browse for Certificate

Certificate file type ☐ PKCS7 ☒ PKCS12

Select Certificate File

Certificate Password

☐ Type or Paste Certificate Contents

29.13.6 What You May Need to Know About the Reports Task Flow

Figure 29–73 shows the unattended tasks report.

Figure 29–73 Unattended Tasks Report

Reports

- Standard Reports
 - Unattended Tasks Report**
 - Tasks Priority Report
 - Tasks Cycle Time Report
 - Tasks Productivity Report
 - Tasks Time Distribution Report

Unattended Tasks Report

Provides an analysis of tasks assigned to users' groups or reportees' groups that need attention because they have not yet been acquired

Assignee

Creation Date to

Expiration Date: to

Task State

Priority

The following worklist reports are available for task analysis.

Unattended Tasks

Unattended Tasks provides an analysis of tasks assigned to users' groups or reportees' groups that have not yet been acquired (the "unattended" tasks).

- **Assignee** -This option (required) selects tasks assigned to the user's group (My Group), tasks assigned to the reportee's groups (Reportees), tasks where the user is a creator (Creator), or tasks where the user is an owner (Owner).
- **Creation Date** - An optional date range
- **Expiration Date** - An optional date range

- **Task State** - The state (optional) can be Any, Assigned, Expired, or Information Requested.
- **Priority** - The priority (optional) can be Any, Highest, High, Normal, Low, or Lowest.

Tasks Priority

Tasks Priority provides an analysis of the number of tasks assigned to a user, reportees, or their groups, broken down by priority.

- **Assignee** - Depending on the assignee that you select, this required option includes tasks assigned to the logged-in user (My), tasks assigned to the user and groups that the user belongs to (My & Group), or tasks assigned to groups to which the user's reportees belong (Reportees).
- **Creation Date** - An optional date range
- **Ended Date** - An optional date range for the end dates of the tasks to be included in the report.
- **Priority** - The priority (optional) can be Any, Highest, High, Normal, Low, or Lowest.

Tasks Cycle Time

Tasks Cycle Time provides an analysis of the time taken to complete tasks from assignment to completion based on users' groups or reportees' groups.

- **Assignee** - Depending on the assignee that you select, this required option includes your tasks (My) or tasks assigned to groups to which your reportees belong (Reportees).
- **Creation Date** - An optional date range
- **Ended Date** - An optional date range for the end dates of the tasks to be included in the report.
- **Priority** - The priority (optional) can be Any, Highest, High, Normal, Low, or Lowest.

Tasks Productivity

Tasks Productivity provides an analysis of assigned tasks and completed tasks in a given time period for a user, reportees, or their groups.

- **Assignee** - Depending on the assignee that the user selects, this required option includes the user's tasks (My & Group) or tasks assigned to groups to which the user's reportees belong (Reportees).
- **Creation Date (range)** - An optional creation date range. The default is one week.
- **Task Type** - Use the Search (flashlight) icon to select from a list of task titles. All versions of a task are listed on the Select Workflow Task Type page (optional).

Tasks Time Distribution

Tasks Time Distribution provides the time an assignee takes to perform a task.

- **Assignee** - Depending on the assignee that the user selects, this required option includes the user's tasks (My & Group) or tasks assigned to groups to which the user's reportees belong (Reportees).
- **From...to (date range)** - An optional creation date range. The default is one week.

- **Task Type** - Use the Search (flashlight) icon to select from a list of task titles. All versions of a task are listed on the Select Workflow Task Type page (optional).

29.13.7 What You May Need to Know About Application Preferences Task Flow

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`. [Figure 29–74](#) provides details.

Figure 29–74 Application Preferences

29.13.8 What You May Need to Know About Mapped Attributes Task Flow

Human workflow mapped attributes store and query use case-specific custom attributes. These custom attributes typically come from the task payload values. Storing custom attributes in mapped attributes provides the following benefits:

- They can be displayed as a column in the task listing.
- They can filter tasks in custom views and advanced searches.
- They can be used for a keyword-based search.

For example the Requester, PurchaseOrderID, and Amount fields in a purchase order request payload of a task can be stored in the mapped attributes. An approver logging into Oracle BPM Worklist can see these fields as column values in the task list and decide which task to access. The user can define views that filter tasks based on the mapped attributes.

For example, a user can create views for purchase order approvals based on different amount ranges. If the user must also retrieve tasks at some point related to a specific requester or a purchase order ID, they can specify this in the keyword field and perform a search to retrieve the relevant tasks. [Figure 29–75](#) provides details.

Figure 29–75 Mapped Attribute Mapping

Administration

Flex Field Mapping

Public Flex Fields

Flex Field Mapping : Public

View existing flex field mappings (i.e. task attribute labels). Create additional flex field labels and map these to task attributes. The labels will be displayed to the end users, and should be user-friendly terms for the task attributes.

☒ Browse all mappings
☐ Edit mappings by task type:

Show: All Attributes

Label	Attribute	Description

Details

Please select a label to see its usage(s).

29.13.9 What You May Need to Know About Rules Task Flow

Rules act on tasks, either a specific task type, or all the tasks assigned to a user or group. The graphic below shows where you set rules, including vacation rules.

A rule cannot always apply in all circumstances in which it is used. For example, if a rule applies to multiple task types, it may not be possible to set the outcome for all tasks, since different tasks can have different outcomes.

Rules are executed in the order in which they are listed. Rules can be reordered by using the up and down buttons in the header. 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.

The `showOtherUsersRules` parameter takes a boolean value. When it is passed as `True` other users' rules are displayed, and when it is passed as `False` other users' rules are not shown. In addition, this user has to have required permission to view other user rules. [Figure 29–76](#) and [Figure 29–77](#) provide details.

Figure 29–76 Vacation Period

Rules

Vacation Period :

Remove yourself from automatic task assignment by enabling a vacation date range. Optionally, more specific vacation rules can be created under "My Rules".

☐ Enable vacation period

Start Date

End Date

Figure 29–77 My Rule

The screenshot displays the 'My Rule' configuration window. On the left, a tree view shows 'Rules' expanded, with 'Vacation Period (Disabled)' and 'My Rules' visible. Under 'My Rules', 'User Rule' is selected. The main area shows the rule configuration:

- Name:** User Rule
- ☐ Use as vacation rule
- Apply only to task type(s):** [Empty field]
- ☐ Execute rule only between these dates:
 - Start Date:** [Empty field]
 - End Date:** [Empty field]
- IF:** Add Condition: Start Date
- THEN:**
 - ☐ Reassign to: [User]
 - ☐ Delegate to: [Empty field]
 - ☐ Set outcome to: [Empty field]
 - ☒ Take no action

At the bottom, a note states: "Reassigned task access is determined according to new assignee rights. Delegated task access is determined according to rights of original user who delegates. 'Take no action' is used to create exception rules that override a more generic rule."

29.14 Java Code for Enabling Customized Applications in Worklist Application

Section 29.8.8, "How to Enable Customized Applications and Links" on page 29-53 explained how to specify a custom application by using the Application Preferences page of Worklist Application. The Java code for performing this specification is as follows:

```
package view.customisationimpl;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import oracle.bpel.services.workflow.client.IWorkflowServiceClient;

import oracle.bpel.services.workflow.runtimeconfig.IRuntimeConfigService;

import oracle.bpel.services.workflow.runtimeconfig.model.AttributeLabelType;

import oracle.bpel.services.workflow.runtimeconfig.model.AttributeLabelUsageList;

import oracle.bpel.services.workflow.runtimeconfig.model.AttributeLabelUsages;

import oracle.bpel.services.workflow.validation.IWorkflowContext;

import oracle.bpm.ui.customization.CustomLink;

import oracle.bpm.ui.customization.IBPMUICustomizations;
```



```
public class WorkspaceCustomisationImpl implements IBPMUICustomizations {

    private static Map displayNameMap = new HashMap();

    public WorkspaceCustomisationImpl() {

        displayNameMap.put("instanceId", "Instance Id");

        displayNameMap.put("protectedTextAttribute1", "Business Status");

    }

    public List<CustomLink> getCustomGlobalLinks() {

        CustomLink globalLink1 =

            new CustomLink("Oracle Home Page", "www.oracle.com", null);

        CustomLink globalLink2 =

            new CustomLink("Self Services Application",
"http://global-ebusiness.oraclecorp.com/",

                null);

        CustomLink globalLink3 =

            new CustomLink("BUG DB", "https://bug.oraclecorp.com/", null);

        List<CustomLink> globalLinks = new ArrayList<CustomLink>();

        globalLinks.add(globalLink1);

        globalLinks.add(globalLink2);

        globalLinks.add(globalLink3);

        return globalLinks;

    }

    public String getColumnNames() {

        return "title,taskNumber,instanceId,creator,protectedTextAttribute1";

    }

    private static void initDisplayMap(IWorkflowServiceClient client,

        IWorkflowContext context) {

        // u can use service to load all label name for text attributes

    }

}
```

```
        if (displayNameMap == null) {

            synchronized (String.class) {

                if (displayNameMap == null) {

                    displayNameMap = new HashMap();

                    try {

                        IRuntimeConfigService service =

                            client.getRuntimeConfigService();

                        AttributeLabelUsageList list =

                            service.getAttributeLabelUsages(context, "Text");

                        List<AttributeLabelUsages> list1 =

                            list.getAttributeLabelUsages();

                        for (AttributeLabelUsages usage : list1) {

                            AttributeLabelType type = usage.getLabel();

                            displayNameMap.put(type.getTaskAttribute(),

                                type.getLabelName());

                        }

                    } catch (Exception exc) {

                    }

                }

            }

        }

    }

    public String getColumnDisplayName(IWorkflowServiceClient client,

                                      IWorkflowContext context,

                                      java.lang.String colName) {

        initDisplayMap(client, context);

        return (String)displayNameMap.get(colName);

    }
```

```
}
```

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 remote EJBs, SOAP, and HTTP.

This chapter includes the following sections:

- [Section 30.1, "Introduction to Building Clients for Workflow Services"](#)
- [Section 30.2, "Packages and Classes for Building Clients"](#)
- [Section 30.3, "Workflow Service Clients"](#)
- [Section 30.4, "Class Paths for Clients Using SOAP"](#)
- [Section 30.5, "Class Paths for Clients Using Remote EJBs"](#)
- [Section 30.6, "Initiating a Task"](#)
- [Section 30.7, "Changing Workflow Standard View Definitions"](#)
- [Section 30.8, "Writing a Worklist Application Using the HelpDeskUI Sample"](#)

30.1 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 30–1](#) 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 30–1 Building a Client for Workflow Services—Setting the Outcome to APPROVE

```
try  
{
```

```
//Create JAVA WorkflowServiceClient
IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(
    WorkflowServiceClientFactory.REMOTE_CLIENT);
//Get the task query service
ITaskQueryService querySvc = wfSvcClient.getTaskQueryService();

//Login as jstein
IWorkflowContext ctx = querySvc.authenticate("jstein","welcome1".toCharArray(),null);
//Set up list of columns to query
List queryColumns = new ArrayList();
queryColumns.add("TASKID");
queryColumns.add("TASKNUMBER");
queryColumns.add("TITLE");
queryColumns.add("OUTCOME");

//Query a list of tasks assigned to jstein
List tasks = querySvc.queryTasks(ctx,
    queryColumns,
    null, //Do not query additional info
    ITaskQueryService.AssignmentFilter.MY,
    null, //No keywords
    null, //No custom predicate
    null, //No special ordering
    0,    //Do not page the query result
    0);
//Get the task service
ITaskService taskSvc = wfSvcClient.getTaskService();
//Loop over the tasks, outputting task information, and approving any
//tasks whose outcome has not been set...
for(int i = 0 ; i < tasks.size() ; i ++){
    Task task = (Task)tasks.get(i);
    int taskNumber = task.getSystemAttributes().getTaskNumber();
    String title = task.getTitle();
    String taskId = task.getSystemAttributes().getTaskId();
    String outcome = task.getSystemAttributes().getOutcome();
    if(outcome == null)
    {
        outcome = "APPROVE";
        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());
}
```

30.2 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 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 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 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 create objects.
- `oracle.bpel.services.workflow.client.IWorkflowServiceClient`
The interface for the workflow service client.
- `oracle.bpel.services.workflow.client.WorkflowServiceClientFactory`
The factory for creating the workflow service client.
- `oracle.bpel.services.workflow.metadata.ITaskMetadataService`
The interface for the task metadata service.
- `oracle.bpel.services.workflow.task.ITaskService`
The interface for the task service.
- `oracle.bpel.services.workflow.task.IRoutingSlipCallback`
The interface for the callback class to receive callbacks during task processing.
- `oracle.bpel.services.workflow.task.IAssignmentService`
The interface for the assignment service.

30.3 Workflow Service Clients

Any worklist application accesses the various workflow services through the workflow service client. The workflow service client code encapsulates all the logic required for communicating with the workflow services using different local and remote protocols. After the worklist application has an instance of the workflow service client, it does not need to consider how the client communicates with the workflow services.

The advantages of using the client are as follows:

- Hides the complexity of the underlying connection mechanisms such as SOAP/HTTP and Enterprise JavaBeans
- Facilitates changing from using one particular invocation mechanism to another, for example from SOAP/HTTP to remote Enterprise JavaBeans

The following class is used to create instances of the `IWorkflowServiceClient` interface:

```
oracle.bpel.services.workflow.client.WorkflowServiceClientFactory
```

`WorkflowServiceClientFactory` has several methods that create workflow clients. The simplest method, `getWorkflowServiceClient`, takes a single parameter, the client type. The client type can be one of the following:

- `WorkflowServiceClientFactory.REMOTE_CLIENT`—The client uses a remote Enterprise JavaBeans interface to invoke workflow services located remotely from the client.
- `WorkflowServiceClientFactory.SOAP_CLIENT`—The client uses SOAP to invoke web service interfaces to the workflow services, located remotely from the client.

The other factory methods enable you to specify the connection properties directly (rather than having the factory load them from the `wf_client_config.xml` file), and enable you to specify a logger to log client activity.

The following enhancements to the workflow service clients are included in this release:

- You can specify the workflow client configuration using either a JAXB object or a map, as shown in [Example 30–2](#) and [Example 30–3](#).

Example 30–2 Workflow Client Configuration Using a JAXB Object

```
WorkflowServicesClientConfigurationType wscct = new WorkflowServicesClientConfigurationType();
List<ServerType> servers = wscct.getServer();
ServerType server = new ServerType();
server.setDefault(true);
server.setName(serverName);
servers.add(server);

RemoteClientType rct = new RemoteClientType();
rct.setServerURL("t3://stapj73:7001");
rct.setUserName("weblogic");
rct.setPassword("weblogic");
rct.setInitialContextFactory("weblogic.jndi.WLInitialContextFactory");
rct.setParticipateInClientTransaction(false);
server.setRemoteClient(rct);
IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(
    WorkflowServiceClientFactory.REMOTE_CLIENT, wscct, logger);
```

Example 30–3 Workflow Client Configuration Using a Map

```
Map<IWorkflowServiceClientConstants.CONNECTION_PROPERTY, java.lang.String> properties = new
    HashMap<IWorkflowServiceClientConstants.CONNECTION_PROPERTY, java.lang.String>();

properties.put(IWorkflowServiceClientConstants.CONNECTION_PROPERTY.MODE,
    IWorkflowServiceClientConstants.MODE_DYNAMIC);

properties.put(IWorkflowServiceClientConstants.CONNECTION_PROPERTY.SOAP_END_POINT_ROOT,
    "http://localhost:8888");

IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory.SOAP_CLIENT,
        properties, null);
```

- Clients can optionally pass in a `java.util.logging.Logger` where the client logs messages. If no logger is specified, then the workflow service client code does not log anything. [Example 30–4](#) shows how a logger can be passed to the workflow service clients.

Example 30–4 Passing a Logger to the Workflow Service Clients

```
java.util.logging.Logger logger = ....;
```



```
IWorkflowServiceClient client =
WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory.REMOTE_CLIENT,
properties, logger);
```

Through the factory, it is possible to get the client libraries for all the workflow services. See [Table 31–1](#), "Enterprise JavaBeans, SOAP, and Java Support" for the clients available for each of the services.

Note that you can obtain instances of `BPMIdentityService` and `BPMIdentityConfigService` by calling the `getSOAPIdentityServiceClient` and `getSOAPIdentityConfigServiceClient` methods on `WorkflowServiceClientFactory`. You can obtain all other services through an instance of `IWorkflowServiceClient`.

The client classes use the configuration file `wf_client_config.xml` for the service endpoints. In the client class path, this file is in the class path directly, meaning the containing directory is in the class path. The `wf_client_config.xml` file contains:

- A section for remote clients, as shown in [Example 30–5](#).

Example 30–5 Section for Remote Clients

```
<remoteClient>
  <serverURL>t3://hostname.domain_name:7001</serverURL>
  <userName>weblogic</userName>
  <password>weblogic</password>
  <initialContextFactory>weblogic.jndi.WLInitialContextFactory
  </initialContextFactory>
  <participateInClientTransaction>>false</participateInClientTransaction>
</remoteClient>
```

- A section for SOAP endpoints for each of the services, as shown in [Example 30–6](#).

Example 30–6 Section for SOAP Endpoints

```
<soapClient>
  <rootEndPointURL>http://hostname.domain_name:7001</rootEndPointURL>
  <identityPropagation mode="dynamic" type="saml">
    <policy-references>
      <policy-reference enabled="true" category="security"
        uri="oracle/wss10_saml_token_client_policy"/>
    </policy-references>
  </identityPropagation>
</soapClient>
```

The workflow client configuration XML schema definition is in the `wf_client_config.xsd` file.

30.3.1 The IWorkflowServiceClient Interface

The `IWorkflowServiceClient` interface provides methods, summarized in [Table 30–1](#), for obtaining handles to the various workflow services interfaces.

Table 30–1 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>

Table 30–1 (Cont.) IWorkflowServiceClient Methods

Method	Interface
getTaskMetadataService	oracle.bpel.services.workflow.metadata.ITaskMetadataService
getUserMetadataService	oracle.bpel.services.workflow.user.IUserMetadataService
getRuntimeConfigService	oracle.bpel.services.workflow.runtimeconfig.IRuntimeConfigService
getTaskEvidenceService	oracle.bpel.services.workflow.metadata.ITaskMetadataService

30.4 Class Paths for Clients Using SOAP

SOAP clients must have the following JAR files in their class path:

- `${bea.home}/wlserver_10.3/server/lib/wlfullclient.jar`
- `${bea.home}/AS11gR1SOA/webservices/wsclient_extended.jar`
- `${bea.home}/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar`
- `${bea.home}/AS11gR1SOA/soa/modules/oracle.soa.workflow_11.1.1/bpm-services.jar`
- `${bea.home}/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/fabric-runtime.jar`

You can generate the `wlfullclient.jar` file using the commands shown in [Example 30–7](#).

Example 30–7 *wlfullclient.jar* File Generation

```
cd ${bea.home}/wlserver_10.3/server/lib
java -jar ../../../../modules/com.bea.core.jarbuilder_1.3.0.0.jar
```

Note: Client applications no longer use the `system\services\config` or `system\services\schema` directories in the class path.

30.5 Class Paths for Clients Using Remote EJBs

Clients using remote EJBs must have the following JAR files in their class path:

- `xmlparserv2.jar`
- `xml.jar`
- `bpm-infra.jar`
- `bpm-services.jar`
- `bpm-services-client.jar` (only if you are using the ADF data controls for workflow)
- `fabric-runtime.jar`

Note: Client applications no longer use the `system\services\config` or `system\services\schema` directories in the class path.

30.6 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`.

30.6.1 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.

30.6.2 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 30–8](#) shows how to set a task payload.

Example 30–8 *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.

30.6.3 Initiating a Task Programmatically

[Example 30–9](#) shows how to initiate a vacation request task programmatically.

Example 30–9 *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. taskDefinitionId is the target
// namespace of the task
// If namespace is used, the active version of the composite corresponding
// to that of the namespace will be used.
task.setTaskDefinitionId("http://xmlns.oracle.com/VacationRequest/
Project1/Humantask1"); (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;
```

30.7 Changing Workflow Standard View Definitions

The worklist application and the `UserMetadataService` API provide methods that you can use to create, update, and delete standard views. See [Section 31.1.7, "User Metadata Service"](#) for more information.

30.8 Writing a Worklist Application Using the HelpDeskUI Sample

The following example shows how to modify the help desk interface that is part of the `HelpDeskRequest` demo.

To write a worklist application

1. Create the workflow context by authenticating the user.

```
// get workflow service client
IWorkflowServiceClient wfSvcClient =
    WorkflowServiceClientFactory.getWorkflowServiceClient
        (WorkflowServiceClientFactory.REMOTE_CLIENT);

//get the workflow context
IWorkflowContext wfCtx =
    wfSvcClient.getTaskQueryService().authenticate(userId, pwd, null);
```

This is Step 3 in [Section 30.1, "Introduction to Building Clients for Workflow Services."](#)

The `login.jsp` file of `HelpDeskRequest` uses the preceding API to authenticate the user and create a workflow context. After the user is authenticated, the `statusPage.jsp` file displays the tasks assigned to the logged-in user.

[Example 30–10](#) shows sample code from the `login.jsp` file.

Example 30–10 Login.jsp

```
<%@ page import="javax.servlet.http.HttpSession"
    import="oracle.bpel.services.workflow.client.IWorkflowServiceClient"
    import="oracle.bpel.services.workflow.client.WorkflowServiceClientFactory"
    import="java.util.Set"
    import="java.util.Iterator"
    import="oracle.bpel.services.workflow.verification.IWorkflowContext"
    import="oracle.tip.pc.services.identity.config.ISConfiguration"%>
<%@ page contentType="text/html; charset=windows-1252"%>

<html>
<head>
<title>Help desk request login page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#F0F0F0" text="#000000" style="font: 12px verdana; line-height:18px">
<center>
<div style="width:640px;padding:15px;border-width: 10px; border-color: #87b4d9; border-style:
```

```
solid;
background-color:white; text-align:left">

<!-- Page Header, Application banner, logo + user status -->
<jsp:include page="banner.jsp"/>

<!-- Initiate Meta Information -->

<div style="background-color:#F0F0F0; border-top:10px solid white;border-bottom:
    10px solid white;padding:10px;text-align:center" >
<b>Welcome to the HelpDesk application</b>
</div>

<%
String redirectPrefix = "/HelpDeskUI/";
// Ask the browser not to cache the page
response.setHeader("Pragma", "no-cache");
response.setHeader("Cache-Control", "no-cache");

HttpSession httpSession = request.getSession(false);
if (httpSession != null) {

    IWorkflowContext ctx = (IWorkflowContext) httpSession.getAttribute("workflowContext");
    if (ctx != null) {
        response.sendRedirect(redirectPrefix + "statusPage.jsp");
    }
    else
    {
        String authFailedStr = request.getParameter("authFailed");
        boolean authFailed = false;
        if ("true".equals(authFailedStr))
        {
            authFailed = true;
        }
        else
        {
            authFailed = false;
        }

        if (!authFailed)
        {
            //Get page parameters:
            String userId="";
            if(request.getParameter("userId") != null)
            {
                userId = request.getParameter("userId");
            }
            String pwd="";
            if(request.getParameter("pwd") != null)
            {
                pwd = request.getParameter("pwd");
            }

            if(userId != null && (!"".equals(userId.trim()))
                && pwd != null && (!"".equals(pwd.trim()))
            {
                try {
                    HttpSession userSession = request.getSession(true);

                    IWorkflowServiceClient wfSvcClient =
```

```

        WorkflowServiceClientFactory.getWorkflowServiceClient
            (WorkflowServiceClientFactory.REMOTE_CLIENT);
        IWorkflowContext wfCtx =
            wfSvcClient.getTaskQueryService().authenticate(userId, pwd, null);
        httpSession.setAttribute("workflowContext", wfCtx);
        response.sendRedirect(redirectPrefix + "statusPage.jsp");
    }
    catch (Exception e)
    {
        String worklistServiceError = e.getMessage();
        response.sendRedirect(redirectPrefix + "login.jsp?authFailed=true");
        out.println("error is " + worklistServiceError);
    }
    }
    else
    {
        out.println("Authentication failed");
    }
}
}
%>

<form action='<%= request.getRequestURI() %>' method="post">
    <div style="width:100%">
        <table cellpadding="2" cellspacing="3" border="0" width="30%" align="center">
            <tr>
                <td>Username
                </td>
                <td>
                    <input type="text" name="userId"/>
                </td>
            </tr>
            <tr>
                <td>Password
                </td>
                <td>
                    <input type="password" name="pwd"/>
                </td>
            </tr>
            <tr>
                <td>
                    <input type="submit" value="Submit"/>
                </td>
            </tr>
        </table>
    </form>
</div>
</div>
</center>
</body>
</html>

```

2. Query tasks using the queryTask API from TaskQueryService.

```

//add list of attributes to be queried from the task
List displayColumns = new ArrayList();
displayColumns.add("TASKNUMBER");
displayColumns.add("TITLE");
displayColumns.add("PRIORITY");
displayColumns.add("STATE");
displayColumns.add("UPDATEDDATE");

```

```

displayColumns.add("UPDATEDBY");
displayColumns.add("CREATOR");
displayColumns.add("OUTCOME");
displayColumns.add("CREATEDDATE");
displayColumns.add("ASSIGNEEUSERS");
displayColumns.add("ASSIGNEEGROUPS");
// get the list of tasks
List tasks = wfSvcClient.getTaskQueryService().queryTasks
    (wfCtx,
     displayColumns,
     null,
     ITaskQueryService.AssignmentFilter.MY_AND_GROUP,
     null,
     null,
     null,
     0,
     0);
// create listing page by using above tasks
//add href links to title to display details of the task by passing taskId
as input parameter
Use getTaskDetailsById(IWorkflowContext wftx, String taskId);

```

This is Step 4 in [Section 30.1, "Introduction to Building Clients for Workflow Services."](#)

The `statusPage.jsp` file of `HelpDeskRequest` is used to display the status of help desk requests. [Example 30–11](#) shows the `statusPage.jsp` example code.

Example 30–11 *statusPage.jsp*

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="oracle.tip.pc.services.identity.BPMAuthorizationService,
    oracle.bpel.services.workflow.verification.IWorkflowContext,
    oracle.tip.pc.services.common.ServiceFactory,
    oracle.bpel.services.workflow.client.IWorkflowServiceClient,
    oracle.bpel.services.workflow.client.WorkflowServiceClientFactory,
    oracle.bpel.services.workflow.query.ITaskQueryService,
    oracle.bpel.services.workflow.task.model.Task,
    oracle.bpel.services.workflow.task.model.IdentityType,
    oracle.tip.pc.services.identity.BPMUser,
    java.util.List,
    java.util.Calendar,
    java.text.SimpleDateFormat,
    java.util.ArrayList"%>
<%@ page contentType="text/html; charset=UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>RequestPage</title>
<style TYPE="text/css">
    Body, Form, Table, Textarea, Select, Input, Option
    {
        font-family : tahoma, verdana, arial, helvetica, sans-serif;
        font-size : 9pt;
    }
    table.banner
    {
        background-color: #eaeff5;
    }
    tr.userInfo

```



```

        {
            background-color: #eaeff5;
        }
        tr.problemInfo
        {
            background-color: #87b4d9;
        }
    </style>
</head>
<body bgcolor="White">
<%
    HttpSession httpSession = request.getSession(false);
    httpSession.setAttribute("pageType", "STATUSPAGE");
%>
<table bordercolor="#eaeff5" border="4" width="100%">
    <tr><td> <jsp:include page="banner.jsp" /> </td></tr>
</table>
<%
    BPMUser bpmUser = null;
    String redirectPrefix = request.getContextPath() + "/";
    IWorkflowContext ctx = null;
    if (httpSession != null) {

        ctx = (IWorkflowContext) httpSession.getAttribute("workflowContext");
        if (ctx != null) {
            bpmUser = getAuthorizationService(ctx.getIdentityContext()).
                lookupUser(ctx.getUser());
        }
        else
        {
            response.sendRedirect(redirectPrefix + "login.jsp");
            return;
        }
    }
    else
    {
        response.sendRedirect(redirectPrefix + "login.jsp");
        return;
    }
    if(bpmUser == null)
    {
        response.sendRedirect(redirectPrefix + "login.jsp");
        return;
    }
    String status = (String)httpSession.getAttribute("requeststatus");
    if(status != null && !status.equals(""))
    {
%>
        <p></p>
        <div style="text-align:left;color:red" >
            <%= status %>
        </div>
<%
    }
    httpSession.setAttribute("requeststatus",null);
    IWorkflowServiceClient wfSvcClient =
        WorkflowServiceClientFactory.getWorkflowServiceClient(
            WorkflowServiceClientFactory.REMOTE_CLIENT);
    List displayColumns = new ArrayList();
    displayColumns.add("TASKNUMBER");

```

```

displayColumns.add("TITLE");
displayColumns.add("PRIORITY");
displayColumns.add("STATE");
displayColumns.add("UPDATEDDATE");
displayColumns.add("UPDATEDBY");
displayColumns.add("CREATOR");
displayColumns.add("OUTCOME");
displayColumns.add("CREATEDDATE");
displayColumns.add("ASSIGNEEUSERS");
displayColumns.add("ASSIGNEEGROUPS");
List tasks = wfSvcClient.getTaskQueryService().queryTasks
    (ctx,
     displayColumns,
     null,
     ITaskQueryService.ASSIGNMENT_FILTER_CREATOR,
     null,
     null,
     null,
     0,
     0);
%>
<p></p>
<div style="text-align:left;color:green" >
  <b>
    Previous help desk request
  </b>
</div>
<p></p>
<div style="text-align:center" >
<table cellpadding="2" cellspacing="2" border="3" width="100%">
  <TR class="problemInfo">
    <TH>TaskNumber</TH>
    <TH>Title</TH>
    <TH>Priority</TH>
    <TH>CreatedDate</TH>
    <TH>Assignee(s)</TH>
    <TH>UpdatedDate</TH>
    <TH>UpdatedBy</TH>
    <TH>State</TH>
    <TH>Status</TH>
  </TR>
<%
  SimpleDateFormat dflong = new SimpleDateFormat( "MM/dd/yy hh:mm a" );
  for(int i = 0 ; i < tasks.size() ; i ++){
    Task task = (Task)tasks.get(i);
    int taskNumber = task.getSystemAttributes().getTaskNumber();
    String title = task.getTitle();
    int priority = task.getPriority();
    String assignee = getAssigneeString(task);
    Calendar createdDate = task.getSystemAttributes().getCreatedDate();
    Calendar updateDate = task.getSystemAttributes().getUpdatedDate();
    String updatedBy = task.getSystemAttributes().getUpdatedBy().getId();
    String state = task.getSystemAttributes().getState();
    String outcome = task.getSystemAttributes().getOutcome();
    if(outcome == null) outcome = "";
    String titleLink = "http://" + request.getServerName() +
      ":" + request.getServerPort() +
      "/integration/worklistapp/TaskDetails?taskId=" +
      task.getSystemAttributes().getTaskId();

```

```

%>
    <tr class="userInfo">
        <td><%=taskNumber%></td>
        <td><a href="<%=titleLink%>" target="_blank"><%=title%></a></td>
        <td><%=priority%></td>
        <td><%=dflong.format(createdDate.getTime())%></td>
        <td><%=assignee%></td>
        <td><%=dflong.format(updateDate.getTime())%></td>
        <td><%=updatedBy%></td>
        <td><%=state%></td>
        <td><%=outcome%></td>
    </tr>
<%
}
%>
</table>
</div>
<%!
private BPMAuthorizationService getAuthorizationService(String identityContext)
{
    BPMAuthorizationService authorizationService =
ServiceFactory.getAuthorizationServiceInstance();
    if (identityContext != null)
        authorizationService = ServiceFactory.getAuthorizationServiceInstance(identityContext);

    return authorizationService;
}
private String getAssigneeString(Task task) throws Exception
{
    List assignees = task.getSystemAttributes().getAssigneeUsers();
    StringBuffer buffer = null;
    for(int i = 0 ; i < assignees.size() ; i++)
    {
        IdentityType type = (IdentityType)assignees.get(i);
        String name = type.getId();
        if(buffer == null)
        {
            buffer = new StringBuffer();
        }
        else
        {
            buffer.append(",");
        }
        buffer.append(name).append(" (U)");
    }
    assignees = task.getSystemAttributes().getAssigneeGroups();
    for(int i = 0 ; i < assignees.size() ; i++)
    {
        IdentityType type = (IdentityType)assignees.get(i);
        String name = type.getId();
        if(buffer == null)
        {
            buffer = new StringBuffer();
        }
        else
        {
            buffer.append(",");
        }
        buffer.append(name).append(" (G)");
    }
}

```

```
        if(buffer == null)
        {
            return "";
        }
        else
        {
            return buffer.toString();
        }
    }
    %>
</body>
</html>
```

Introduction to Human Workflow Services

This chapter describes how the human workflow services are used. These services perform a variety of operations in the life cycle of a task.

This chapter includes the following sections:

- [Section 31.1, "Introduction to Human Workflow Services"](#)
- [Section 31.2, "Notifications from Human Workflow"](#)
- [Section 31.3, "Assignment Service Configuration"](#)
- [Section 31.4, "Class Loading for Callbacks and Resource Bundles"](#)
- [Section 31.5, "Resource Bundles in Workflow Services"](#)
- [Section 31.6, "Introduction to Human Workflow Client Integration with Oracle WebLogic Server Services"](#)
- [Section 31.7, "Task States in a Human Task"](#)
- [Section 31.8, "Database Views for Oracle Workflow"](#)

Note: In previous releases, Oracle BPM Worklist included a feature known as flex fields. Starting with Release 11g R1 (11.1.1.4), flex fields are now known as mapped attributes.

31.1 Introduction to Human Workflow Services

This section describes the responsibilities of the following human workflow services.

- Task service
- Task query service
- Identity service
- Task metadata service
- User metadata service
- Task report service
- Runtime config service
- Evidence store service

31.1.1 SOAP, Enterprise JavaBeans, and Java Support for the Human Workflow Services

Table 31–1 lists the type of Simple Object Access Protocol (SOAP), Enterprise JavaBeans, and Java support provided for the task services. Most human workflow services are accessible through SOAP and remote Enterprise JavaBeans APIs. You can use these services directly by using appropriate client proxies. Additionally, the client libraries are provided to abstract out the protocol details and provide a common interface for all transports.

Table 31–1 Enterprise JavaBeans, SOAP, and Java Support

Service Name	Supports SOAP Web Services	Supports Remote Enterprise JavaBeans
Task Service: Provides task state management and persistence of tasks. In addition to these services, the task service exposes operations to update a task, complete a task, escalate and reassign tasks, and so on.	Yes	Yes
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
Identity Service: Enables authentication of users and the lookup of user properties, roles, group memberships, and privileges.	Yes	No
Task Metadata Service: Exposes operations to retrieve metadata information related to a task.	Yes	Yes
User Metadata Service: Manages metadata related to workflow users, such as user work queues, preferences, vacation, and delegation rules.	Yes	Yes
Task Reports Service: Provides workflow report details.	Yes	Yes
Runtime Config Service: Provides methods for managing metadata used in the task service runtime environment.	Yes	Yes
Evidence Store Service: Supports storage and nonrepudiation of digitally-signed workflow tasks.	Yes	Yes

Table 31–2 lists the location for the SOAP Web Services Description Language (WSDL) file for each task service.

Table 31–2 SOAP WSDL Location for the Task Services

Service name	SOAP WSDL location
Task Service	<code>http://host:port/integration/services/TaskService/TaskServicePort?WSDL</code>
Task Query Service	<code>http://host:port/integration/services/TaskQueryService/TaskQueryService?WSDL</code>
Identity Service	<code>http://host:port/integration/services/IdentityService/configuration?WSDL</code> <code>http://host:port/integration/services/IdentityService/identity?WSDL</code>
Task Metadata Service	<code>http://host:port/integration/services/TaskMetadataService/TaskMetadataServicePort?WSDL</code>

Table 31–2 (Cont.) SOAP WSDL Location for the Task Services

Service name	SOAP WSDL location
User Metadata Service	<code>http://host:port/integration/services/UserMetadataService/UserMetadataService?WSDL</code>
Task Report Service	<code>http://host:port/integration/services/TaskReportService/TaskReportServicePort?WSDL</code>
Runtime Config Service	<code>http://host:port/integration/services/RuntimeConfigService/RuntimeConfigService?WSDL</code>
Evidence Store Service	<code>http://host:port/integration/services/EvidenceService/EvidenceService?WSDL</code>

Table 31–3 lists the JNDI names for the different Enterprise JavaBeans.

Table 31–3 JNDI Names for the Different Enterprise JavaBeans

Service name	JNDI Names for the Different Enterprise JavaBeans
Task Service	<code>ejb/bpel/services/workflow/TaskServiceBean</code>
Task Service Enterprise JavaBeans participating in client transaction	<code>ejb/bpel/services/workflow/TaskServiceGlobalTransactionBean</code>
Task Metadata Service	<code>ejb/bpel/services/workflow/TaskMetadataServiceBean</code>
Task Query Service	<code>ejb/bpel/services/workflow/TaskQueryService</code>
User Metadata Service	<code>ejb/bpel/services/workflow/UserMetadataService</code>
Runtime Config Service	<code>ejb/bpel/services/workflow/RuntimeConfigService</code>
Task Report Service	<code>ejb/bpel/services/workflow/TaskReportServiceBean</code>
Task Evidence Service	<code>ejb/bpel/services/workflow/TaskEvidenceServiceBean</code>

For more information about the client library for worklist services, see [Chapter 30](#), "Building a Custom Worklist Client" for details.

31.1.1.1 Support for Foreign JNDI Names

Human workflow services can be integrated with J2EE applications through web services and remote method invocation (RMI). To simplify the remote lookup of Enterprise JavaBeans in other managed servers and clusters or even other Oracle WebLogic Server domains, Oracle WebLogic Server includes foreign JNDI providers that are configured with the remote server's host and port to link Enterprise JavaBeans from that remote server into the local server's JNDI trees.

Workflow services expose the Enterprise JavaBeans listed in [Table 31–3](#) that must all be linked through the foreign JNDI providers to provide full support for the task query service, ADF task flow for human task registration, and embedded worklist region use cases.

To provide support for foreign JNDI names:

1. Log in to Oracle WebLogic Server Administration Console.

`http://host:port/console`

2. In the **Domain Structure**, select **Services > JDBC > Foreign JNDI Providers**.

There is one caveat when linking remote Enterprise JavaBeans names to the local JNDI namespace through a foreign JNDI provider from a SOA server to a managed server or cluster in the same Oracle WebLogic Server domain. The local JNDI names are exposed to all of the managed servers within that domain. This causes namespace collisions on the SOA server within that domain, which already has those Enterprise JavaBeans registered from the Oracle BPM Worklist. An alternative, which avoids collisions while keeping configuration to a minimum, is to use JNDI suffixing. This is done by appending a consistent suffix to the end of all the local JNDI links of the remote workflow Enterprise JavaBeans and creating a simple `wf_client_config.xml` file that contains the suffix key.

You can define client properties in either of three ways. For more information, see [Section 31.6.1.2, "Configuration Option."](#)

3. Append the JNDI suffix to each Enterprise JavaBeans name shown in [Table 31–3](#) to register the foreign JNDI names.
 - `ejb/bpel/services/workflow/TaskServiceGlobalTransactionean_server1`
 - `ejb/bpel/services/workflow/TaskServiceBean_server1`
 - `ejb/bpel/services/workflow/TaskMetadataServiceBean_server1`
 - `TaskQueryService_server1`
 - `UserMetadataService_server1`
 - `RuntimeConfigService_server1`
 - `TaskReportServiceBean_server1`
 - `TaskEvidenceServiceBean_server1`
4. Define the remote name by specifying only the `ejbJndiSuffix` element value in the `wf_client_config.xml` file, as shown in [Example 31–1](#). You can also use the `JAXB WorkflowServicesClientConfigurationType` object or the `CONNECTION_PROPERTY.EJB_JNDI_SUFFIX` in the `Map<CONNECTION_PROPERTY, String>` properties.

Example 31–1 *ejbJndiSuffix Element Value*

```
<remoteClient>
  <ejbJndiSuffix>_server1</ejbJndiSuffix>
</remoteClient>
```

31.1.2 Security Model for Services

With the exception of the identity service, all services that use the above-mentioned APIs (SOAP and remote Enterprise JavaBeans) require authentication to be invoked. All the above channels support passing the user identity using the human workflow context. The human workflow context contains either of the following:

- Login and password
- Token

The task query service exposes the `authenticate` operation that takes the login and password and returns the human workflow context used for all services. Optionally, with each request, you can pass the human workflow context with the login and password.

The `authenticate` operation also supports the concept of creating the context on behalf of a user with the admin ID and admin password. This operation enables you to create the context for a logged-in user to the Oracle BPM Worklist if the password for that user is not available.

Oracle recommends that you get the workflow context one time and use it everywhere. There are performance implications for getting the workflow context for every request.

A realm is an identity service context from the identity configuration. The realm name can be null if the default configuration is used.

31.1.2.1 Limitation on Propagating Identity to Workflow Services when Using SOAP Web Services

Identity propagation is the replication of authenticated identities across multiple SOAP web services used to complete a single transaction. SOAP web services also support web service security. When web service security is used, the human workflow context does not need to be present in the SOAP input. Web service security can be configured from Oracle Enterprise Manager Fusion Middleware Control.

Note: Human workflow SOAP clients have been enhanced to work with Security Assertion Markup Language (SAML) token-based identity propagation when the web service is secured.

31.1.2.2 Creating Human Workflow Context on Behalf of a User

The `authenticateOnBehalfOf` API method on the task query service can create the human workflow context on behalf of a user by passing the user ID and password of an admin user in the request. An admin user is a user with the `workflow.admin` privilege. This created context is as if it was created using the password on behalf of the user.

This is useful for environments in which a back-end system acts on workflow tasks while users act in their own system. There is no direct interaction with workflow services; the system can use the on-behalf-of-user login to get a context for the user.

Note: Oracle recommends that you only use this feature for system operations. This is because you must create an admin user context and then query for the human workflow context created on behalf of the user. If you instead use identity propagation, the user is already authenticated and the client can get `IWorkflowContext` for the already authenticated user. For more information, see [Section 31.1.2.3, "Obtaining the Workflow Context for a User Previously Authenticated by a JAAS Application."](#)

In [Example 31-2](#), the human workflow context is created for user `jcooper`.

Example 31-2 Human Workflow Context Creation

```
String adminUser = "...."
String adminPassword = "...."
String realm = "...."

IWorkflowContext adminCtx =
taskQueryService.authenticate(user,password.toCharArray(),realm);
```

```
IWorkflowContext behalfOfCtx =
    taskQueryService.authenticateOnBehalfOf(adminCtx, "jcooper");
```

31.1.2.3 Obtaining the Workflow Context for a User Previously Authenticated by a JAAS Application

If the client wants to obtain the workflow context for a user previously authenticated by a JAAS application, you can use identity propagation as shown in [Example 31–3](#).

Example 31–3 Identity Propagation

```
public IWorkflowContext getWorkflowContextForAuthenticatedUser() throws
WorkflowException;
```

This API returns a workflow context for the authenticated user if the client configures the identity propagation for the appropriate client type. If the client type is remote, Enterprise JavaBeans identity propagation is used with this method. If the client type is SOAP, SAML token propagation is used with this method.

31.1.3 Task Service

The task service exposes operations to act on tasks. [Table 31–4](#) describes the operations of the task service. Package `oracle.bpel.services.workflow.task` corresponds to the task service.

Table 31–4 Task Service Methods

Method	Description
<code>acquireTask</code>	Acquire a task.
<code>acquireTasks</code>	Acquire a set of tasks.
<code>addAttachment</code>	Add an attachment to a task.
<code>addComment</code>	Add a comment to a task.
<code>createToDoTask</code>	Create a to-do task.
<code>delegateTask</code>	Delegate a task to a different user. Both the current assignee and the user to whom the task is delegated can view and act on the task.
<code>delegateTasks</code>	Delegate a list of tasks to a different user. Both the current assignee and the user to whom the list of tasks is delegated can view and act on the tasks.
<code>deleteTask</code>	Perform a logical deletion of a task. The task still exists in the database.
<code>deleteTasks</code>	Perform a logical deletion of a list of tasks. The tasks still exist in the database.
<code>errorTask</code>	Cause the task to error. This operation is typically used by the error assignee.
<code>escalateTask</code>	Escalate a task. The default escalation is to the manager of the current user. This can be overridden using escalation functions.
<code>escalateTasks</code>	Escalate tasks in bulk. The default escalation is to the manager of the current user. This can be overridden using escalation functions.
<code>getApprovers</code>	Get the previous approvers of a task.

Table 31–4 (Cont.) Task Service Methods

Method	Description
<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).
<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: ■ 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 PushbackAssignee in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control controls whether the task is pushed back to the original assignees or the approvers. 1. From the SOA Infrastructure menu, select Administration > System MBean Browser. 2. Select Application Defined MBeans > oracle.as.soainfra.config > Server: soa_server1 > WorkflowConfig > human-workflow. 3. Click PushbackAssignee to view or change the value.
<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.
<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.

Table 31–4 (Cont.) Task Service Methods

Method	Description
<code>requestInfoForTaskWithReapproval</code>	Request information for a task with reapproval. For example, assume <code>jcooper</code> created a task and <code>jstein</code> and <code>wfaulk</code> approved the task in the same order. When the next approver, <code>cdickens</code> , requests information with reapproval from <code>jcooper</code> , and <code>jcooper</code> submits the information, <code>jstein</code> and <code>wfaulk</code> approve the task before it comes to <code>cdickens</code> . If <code>cdickens</code> requests information with reapproval from <code>jstein</code> , and <code>jstein</code> submits the information, <code>wfaulk</code> approves the task before it comes to <code>cdickens</code> .
<code>resumeTask</code>	Resume a task. Operations can only be performed by the task owners (or users with the <code>BPMWorkflowSuspend</code> privilege) to remove the hold on a workflow. After a human workflow is resumed, actions can be performed on the task.
<code>resumeTasks</code>	Resume a set of tasks.
<code>routeTask</code>	Allow a user to route the task in an ad hoc fashion to the next user(s) who must review the task. The user can specify to route the tasks in serial, parallel, or single assignment. Routing a task is permitted only when the human workflow permits ad hoc routing of the task.
<code>skipCurrentAssignment</code>	Skip the current assignment and move to the next assignment or pick the outcome as set by the previous approver if there are no more assignees.
<code>submitInfoForTask</code>	Submit information for a task. This action is typically performed after the user has made the necessary updates to the task or has added comments or attachments containing additional information.
<code>suspendTask</code>	Allow task owners (or users with the <code>BPMWorkflowSuspend</code> privilege) to put a human workflow on hold temporarily. In this case, task expiration and escalation do not apply until the workflow is resumed. No actions are permitted on a task that has been suspended (except resume and withdraw).
<code>suspendTasks</code>	Suspend a set of tasks.
<code>updateOutcomeOfTasks</code>	Update the outcome of a set of tasks.
<code>updatePriority</code>	Update the priority of the task and its subtasks for the given <code>taskId</code> . If <code>UpdatePriorityType</code> is <code>INCREMENT</code> then the task will be updated by incrementing the given priority by 1—that is, the priority of the task will be raised. If the <code>UpdatePriorityType</code> is <code>DECREMENT</code> then the task will be updated by decrementing the priority by 1—that is, the priority of the task will be lowered, otherwise the task will be updated with the given priority.
<code>updatePriorityOfTasks</code>	For bulk update of tasks. A list of tasks for which priority needs to be updated can be passed as a parameter to this API. The priorities of the list of tasks is updated. It updates the priority of the task and its subtasks.
<code>updateTask</code>	Update the task.
<code>updateTaskOutcome</code>	Update the task outcome.
<code>updateTaskOutcomeAndRoute</code>	Update the task outcome and route the task. Routing a task allows a user to route the task in an ad hoc fashion to the next user(s) who must review the task. The user can specify to route the tasks in serial, parallel, or single assignment. Routing a task is permitted only when the human workflow permits ad hoc routing of the task.

Table 31–4 (Cont.) Task Service Methods

Method	Description
<code>withdrawTask</code>	The creator of the task can withdraw any pending task if they are no longer interested in sending it further through the human workflow. A task owner can also withdraw a task on behalf of the creator. When a task is withdrawn, the business process is called back with the state attribute of the task set to <code>Withdrawn</code> .
<code>withdrawTasks</code>	Withdraw a set of tasks.

For more information, see the following:

- [Section 31.1.11, "Task Instance Attributes"](#)
- *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*
- Sample workflow-118-JavaSamples, which demonstrates some APIs

31.1.4 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, historical information of a task, and so on. [Table 31–5](#) describes the operations of the task query service, including how to use the service over SOAP. Package `oracle.bpel.services.workflow.query` corresponds to the task query service.

Table 31–5 Task Query Service Methods

Method	Description
<code>authenticate</code>	Authenticates a user with the identity authentication service and passes back a valid <code>IWorkflowContext</code> object.
<code>authenticateOnBehalfOf</code>	Optionally makes authentication on behalf of another user.
<code>countTasks</code>	Counts the number of tasks that match the specified query criteria.
<code>countViewTasks</code>	Counts the number of tasks that match the query criteria of the specified view.
<code>createContext</code>	Creates a valid <code>IWorkflowContext</code> object from a preauthenticated HTTP request.
<code>doesTaskExist</code>	Checks to see if any existing tasks match the specified query criteria.
<code>doesViewTaskExist</code>	Checks to see if any tasks exist match the query criteria of the specified view.
<code>getWorkflowContext</code>	Gets a human workflow context with the specified context token.
<code>destroyWorkflowContext</code>	Cleans up a human workflow context that is no longer needed. This method is typically used when a user logs out.
<code>getTaskDetailsById</code>	Gets the details of a specific task from the task's <code>taskId</code> property.
<code>getTaskDetailsByNumber</code>	Gets the details of a specific task from the task's <code>task number</code> property.
<code>getTaskHistory</code>	Gets a list of the task versions for the specified task ID.

Table 31–5 (Cont.) Task Query Service Methods

Method	Description
<code>getTaskSequence</code>	Gets the task sequence tree of a task whose ID is a task ID, for those type of sequences.
<code>getTaskVersionDetails</code>	Gets the specific task version details for the specified task ID and version number.
<code>getWorkflowContextForAuthenticatedUser</code>	Gets the <code>IWorkflowContext</code> object for a user authenticated by a JAAS application. Use this either with Enterprise JavaBeans or SAML token identity propagation.
<code>queryAggregatedTasks</code>	Executes the specified query, and aggregates a count of the tasks returned by the query, grouped by the specified column.
<code>queryTaskErrors</code>	Returns a list of task error objects matching the specified predicate.
<code>queryTasks</code>	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: ■ <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: ADMIN: No filtering; returns all tasks regardless of assignment or creator. ALL: No filtering; returns all tasks regardless of assignment or creator. CREATOR: Returns tasks in which the context user is the creator. GROUP: Returns tasks that are assigned to a group, application role, or list of users of which the context user is a member. MY: Returns tasks that are assigned exclusively to the context user. MY_AND_GROUP: Returns tasks that are assigned exclusively to the context user, or to a group, application role, or list of users of which the context user is a member. OWNER: Returns tasks in which the context user is the task owner. PREVIOUS: Returns tasks the context user previously updated. REPORTTEES: Returns tasks that are assigned to reportees of the context user. REVIEWER: Returns tasks for which the context user is a reviewer. ■ <i>keywords</i>: An optional search string. This only returns tasks in which the string is contained in the task title, task identification key, or one of the task text mapped attributes (formerly referred to as 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.

Table 31–5 (Cont.) Task Query Service Methods

Method	Description
<code>queryViewAggregatedTasks</code>	Executes the query as defined in the specified view, and aggregates the selected tasks according to the chart property defined in the view.
<code>queryViewTasks</code>	Returns a list of tasks according to the criteria in the specified view. The entire list or paged list of tasks can be returned. Clients can specify additional filter and ordering criteria to those in the view.

For more information, see the following:

- [Section 31.1.11, "Task Instance Attributes"](#)
- *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager* in the documentation library
- Sample workflow-118-JavaSamples, which demonstrates some APIs

31.1.5 Identity Service

The identity service is a thin web service layer on top of the Oracle WebLogic Server security infrastructure, namely Oracle Identity Management and Oracle Platform Security Services (OPSS), or any custom user repository. The identity service enables authentication of users and the lookup of user properties, roles, group memberships, and privileges. Oracle Identity Management is the sole identity service provider for Oracle WebLogic Server. Oracle Identity Management handles all storage and retrieval of users and roles for various repositories, including XML, LDAP, and so on. More specifically, Oracle Identity Management provides the following features:

- All providers are supported through Oracle Identity Management. The OracleAS JAAS Provider (JAZN) and LDAP providers are no longer supported. The custom provider is deprecated and supported only for backward compatibility. All customization of providers is performed through the custom provider to Oracle Identity Management, through configuring Oracle Virtual Directory (OVD) as an LDAP provider to Oracle Identity Management, or through both. OVD aggregates data across various repositories.
- The OPSS layer is used, which includes the following:
 - Identity store
 - Policy store
 - Credential store
 - Framework

For more information, see *Oracle Fusion Middleware Security Guide*. All security configuration is done through the `jps-config.xml` file.

- All privileges are validated against permissions, as compared to actions in previous releases.
- The following set of application roles are defined. These roles are automatically defined in the SOA Infrastructure application of the OPSS policy store.
 - `SOAAdmin`: Grant this role to users who must perform administrative actions on any SOA module. This role is also granted the `BPMWorkflowAdmin` and `B2BAdmin` roles.

- **BPMWorkflowAdmin:** Grant this role to users who must perform any workflow administrative action. This includes actions such as searching and acting on any task in the system, creating and modifying user and group rules, performing application customization, and so on. This role is granted the `BPMWorkflowCustomize` role and the following permissions:
 - * `workflow.mapping.protectedFlexField`
 - * `workflow.admin.evidenceStore`
 - * `workflow.admin`
- **BPMWorkflowCustomize:** Grant this role to business users who must perform mapped attributes (formally flex field) mapping to public mapped attributes. 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 mapped attributes.
 - `workflow.mapping.protectedFlexField:` Controls who can perform mapping of task payload attributes to protected mapped attributes.

Note: You cannot specify multiple authentication providers for Oracle SOA Suite. This is because OPSS does not support multiple providers. The provider to use for human workflow authentication must be the first one listed in the order of authentication providers for Oracle SOA Suite.

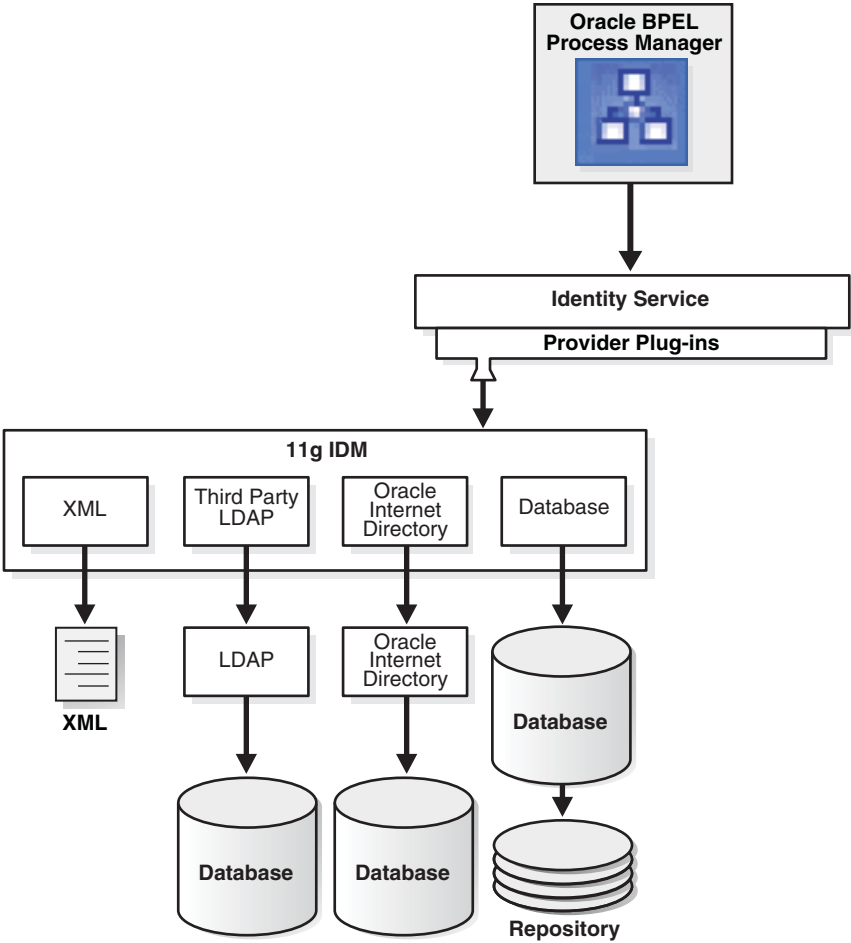
For more information, see the following:

- *Oracle Fusion Middleware Security Guide* for details about OPSS
- *Oracle Fusion Middleware Application Developer's Guide for Oracle Identity Management* for details about Oracle Identity Management
- *Oracle Fusion Middleware Administrator's Guide for Oracle Virtual Directory* for details about OVD

31.1.5.1 Identity Service Providers

Oracle Identity Management is the only supported provider for release 11g, as shown in [Figure 31-1](#).

Figure 31–1 Identity Service Providers



31.1.5.1.1 Custom User Repository Plug-ins Starting with release 11g, custom provider plug-ins in the identity service are not supported. All identity customizations are now done in the identity store. Oracle Fusion Middleware supports providers that enable the User and Role API to interact with custom identity stores. For more information, visit the following URL:

http://www.oracle.com/technology/products/id_mgmt/opss/index.html

31.1.6 Task Metadata Service

The task metadata service exposes operations to retrieve metadata information related to a task. Table 31–6 describes these methods. Package `oracle.bpel.services.workflow.metadata` corresponds to the task metadata service.

Table 31–6 Task Metadata Service Methods

Method	Description
<code>getTaskMetadataByName space</code>	Gets the <code>TaskMetadata</code> object that describes the human task service component with the specified task definition namespace and composite version.
<code>getOutcomes</code>	Gets the permitted outcomes of a task. The outcomes are returned with their display values.

Table 31–6 (Cont.) Task Metadata Service Methods

Method	Description
<code>getResourceBundleInfo</code>	Gets the resource bundle information of the task. The resource bundle information contains the location and the name of the bundle.
<code>getRestrictedActions</code>	Gets the actions that are restricted for a particular task.
<code>getTaskAttributesForTaskDefinitions</code>	Gets a list of <code>TaskAttribute</code> objects that describe standard task attributes and mapped attribute columns that are common for the specified task definitions.
<code>getTaskAttributesForTaskNamespaces</code>	Gets a list of <code>TaskAttribute</code> objects that describe standard task attributes and mapped attribute columns that are common for task definitions identified by the specified namespaces.
<code>getTaskAttributes</code>	Gets the task message attributes.
<code>getTaskAttributesForTaskDefinition</code>	Gets the message attributes for a particular task definition.
<code>getTaskDefinition</code>	Gets the task definition associated with the task.
<code>getTaskDefinitionById</code>	Gets the task definition by the task definition ID.
<code>getTaskDefinitionOutcome</code>	Gets the outcomes given the task definition ID.
<code>getTaskDisplay</code>	Gets the task display for a task.
<code>getTaskVisibilityRules</code>	Gets the task visibility rules.
<code>getTaskDisplayRegion</code>	Gets the task display region for a task.
<code>getVersionTrackedAttributes</code>	Gets the task attributes that when changed cause a task version creation.
<code>listTaskMetadata</code>	Lists the task definitions in the system.

For more information, see *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*.

31.1.7 User Metadata Service

The user metadata service provides methods for managing metadata specific to individual users and groups. It is used for getting and setting user worklist preferences, managing user custom views, and managing human workflow rules for users and groups.

For most methods in the user metadata service, the authenticated user can query and update their own user metadata. However, they cannot update metadata belonging to other users.

In the case of group metadata (for example, human workflow rules for groups), only a user designated as an owner of a group (or a user with the `workflow.admin` privilege) can query and update the metadata for that group. However, a user with the `workflow.admin` privilege can query and update metadata for any user or group.

[Table 31–7](#) describes the methods in the user metadata service. Package `oracle.bpel.services.workflow.user` corresponds to the user metadata service.

Table 31–7 User Metadata Service Methods

Method	Description
<code>createRule</code>	Creates a new rule.
<code>decreaseRulePriority</code>	Decreases the priority of a rule by one. This method does nothing if this rule has the lowest priority.
<code>deleteRule</code>	Deletes a rule.
<code>getVacationInfo</code>	Retrieves the date range (if any) during which a user is unavailable for the assignment of tasks.
<code>getRuleDetail</code>	Gets the details for a particular human workflow rule.
<code>getRuleList</code>	Retrieves a list of rules for a particular user or group.
<code>updateRule</code>	Updates an existing rule.
<code>increaseRulePriority</code>	Increases the priority of a rule by one. Rules for a user or group are maintained in an ordered list of priority. Higher priority rules (those closer to the head of the list) are executed before rules with lower priority. This method does nothing if this rule has the highest priority.
<code>getUserTaskViewList</code>	Gets a list of the user task views that the user owns.
<code>getGrantedTaskViewList</code>	Gets a list of user task views that have been granted to the user by other users. Users can use granted views for querying lists of tasks, but they cannot update the view definition.
<code>getStandardTaskViewList</code>	Gets a list of standard task views that ship with the human workflow service, and are available to all users.
<code>getUserTaskViewDetails</code>	Gets the details for a single view.
<code>createUserTaskView</code>	Creates a new user task view.
<code>updateUserTaskView</code>	Updates an existing user task view.
<code>deleteUserTaskView</code>	Deletes a user task view.
<code>updateGrantedTaskView</code>	Updates details of a view grant made to this user by another user. Updates are limited to hiding or unhiding the view grant (hiding a view means that the view is not listed in the main inbox page of Oracle BPM Worklist), and changing the name and description that the granted user sees for the view.
<code>getUserPreferences</code>	Gets a list of user preferences for the user. User preferences are simple name-value pairs of strings. User preferences are private to each user (but can still be queried and updated by a user with the <code>workflow.admin</code> privilege).
<code>setUserPreferences</code>	Sets the user preference values for the user. Any preferences that were previously stored and are not in the new list of user preferences are deleted.
<code>getPublicPreferences</code>	Gets a list of public preferences for the user. Public preferences are similar to user preferences, except that any user can query them. However, only the user that owns the preferences, or a user with the <code>workflow.admin</code> privilege, can update them. Public preferences are useful for storing application-wide preferences (preferences can be stored under a dummy user name, such as <code>MyAppPrefs</code>).
<code>setPublicPreferences</code>	Sets the public preferences for the user.

Table 31–7 (Cont.) 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>getStandardTaskViewDetails</code>	Gets the full details for a particular standard view, identified by its <code>viewId</code> .

For more information, see the following:

- [Chapter 29, "Using Oracle BPM Worklist"](#) for details about the rule configuration and user preference pages
- `Sample workflow-118-JavaSamples`, which demonstrates some APIs
- *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*

31.1.8 Task Report Service

The task report service executes a report and receives the results. [Table 31–8](#) describes the method. Package `oracle.bpel.services.workflow.report` corresponds to the task report service. The standard reports shown in [Table 31–8](#) are available as part of installation.

Table 31–8 Task Report Service

Report	Description
Unattended tasks report	Provides an analysis of tasks assigned to users' groups or reportees' groups that require attention because they have not yet been acquired.
Tasks priority report	Provides an analysis of the number of tasks by priorities assigned to a user, reportees, or their groups.
Tasks cycle time report	Provides an analysis of time taken to complete tasks from assignment to completion based on users' groups or reportees' groups.
Tasks productivity report	Provides an analysis of tasks assigned and tasks completed in a given time period for a user, reportees, or their groups.
Tasks time distribution report	Provides an analysis of time taken to complete their part of the tasks for a given user, user's groups, or reportees in the given time period.

31.1.9 Runtime Config Service

The runtime config service provides methods for managing metadata used in the task service runtime environment. It principally supports the management of task payload mapped attribute mappings and the URIs used for displaying task details.

The task object used by the task service contains many mapped attributes, which can be populated with information from the task payload. This allows the task payload information to be queried, displayed in task listings, and used in human workflow rules.

The runtime config service provides methods for querying and updating the URI used for displaying the task details of instances of a particular task definition in a client application. For any given task definition, multiple display URIs can be supported,

with different URIs being used for different applications. The method `getTaskDisplayInfo` can query the URIs for a particular task definition. The method `setTaskDisplayInfo` can define new URIs or update existing ones. Only users with the `workflow.admin` privilege can call `setTaskDisplayInfo`, but any authenticated user can call `getTaskDisplayInfo`.

The runtime config service allows administrators to create mappings between simple task payload attributes and these mapped attributes.

Only a user with the `workflow.mapping.publicFlexField` or `workflow.mapping.protectedFlexField` privilege can make updates to payload mappings for public mapped attributes. Only a user with the `workflow.mapping.protectedFlexField` privilege can make updates to payload mappings for protected mapped attributes. Any authenticated user can use the query methods in this service.

An administrator can create attribute labels for the various mapped attributes. These attribute labels provide a meaningful label for the attribute (for example, a label `Location` may be created for the mapped attribute `TextAttribute1`). A given mapped attribute may have multiple labels associated with it. This attribute label is what is displayed to users when displaying lists of attributes for a specific task in Oracle BPM Worklist. The attribute labels for a specific task type can be determined by calling the `getTaskAttributesForTaskDefinition` method on the task metadata service.

When defining attribute labels, the following fields are automatically populated by the service. You do not need to specify values for these attributes when creating or updating attribute labels:

- `Id`
- `CreatedDate`
- `WorkflowType`
- `Active`

Valid values for the task attribute field for public mapped attributes are as follows:

- `TextAttribute1` through `TextAttribute20`
- `FormAttribute1` through `FormAttribute10`
- `UrlAttribute1` through `UrlAttribute10`
- `DateAttribute1` through `DateAttribute10`
- `NumberAttribute1` through `NumberAttribute10`

Mappings can then be created between task payload fields and the attribute labels. For example, the payload field `customerLocation` can be mapped to the attribute label `Location`. Different task types can share the same attribute label. This allows payload attributes from different task types that have the same semantic meaning to be mapped to the same attribute label.

Note: Only payload fields that are simple XML types can be mapped.

The runtime config service also provides the following:

- Methods for querying the dynamic assignment functions supported by the server

- Methods for maintaining the task display URLs used for displaying the task details in Oracle BPM Worklist and other applications
- Methods for getting the server HTTP and JNDI URLs

Table 31–9 describes the methods in the runtime config service. Package `oracle.bpel.services.workflow.runtimeconfig` corresponds to the runtime config service.

Table 31–9 Runtime Config Service

Method	Description
<code>CreateAttributeLabel</code>	Creates a new attribute label for a particular task mapped 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 mapped attribute mappings for a particular human workflow definition.
<code>setTaskDisplayInfo</code>	Sets information relating to the URIs to be used for displaying task instances of a specific task definition.
<code>updateAttributeLabel</code>	Updates an existing attribute label.

For more information, see the following:

- [Section 31.3.1, "Dynamic Assignment and Task Escalation Functions"](#) for additional details
- [Chapter 29, "Using Oracle BPM Worklist"](#) for details about mapped attribute mappings
- Sample workflow-118-JavaSamples, which demonstrates some APIs.
- *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*

31.1.9.1 Internationalization of Attribute Labels

Attribute labels provide a method of attaching a meaningful label to a task mapped attribute. It can be desirable to present attribute labels that are translated into the appropriate language for the locale of the user.

To use a custom resource bundle, place it at the location identified by the workflow configuration parameter `workflowCustomClasspathURL` (which can be a file or HTTP path).

This can be set in either of two places in Oracle Enterprise Manager Fusion Middleware Control:

- System MBean Browser page
- Workflow Task Service Properties page

For more information, see the `workflow-110-workflowCustomizations` sample, which describes how to use this parameter. Visit the following URL for details:

<https://soasamples.samplecode.oracle.com/>

Entries for mapped 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.

31.1.10 Evidence Store Service and Digital Signatures

The evidence store service is used for digital signature storage and nonrepudiation of digitally-signed human workflows. A digital signature is an electronic signature that authenticates the identity of a message sender or document signer. This ensures that the original content of the message or document sent is unchanged. Digital signatures are transportable, cannot be imitated by others, and are automatically time-stamped. The ability to ensure that the original signed message arrived means that the sender cannot repudiate it later. Digital signatures ensure that a human workflow document:

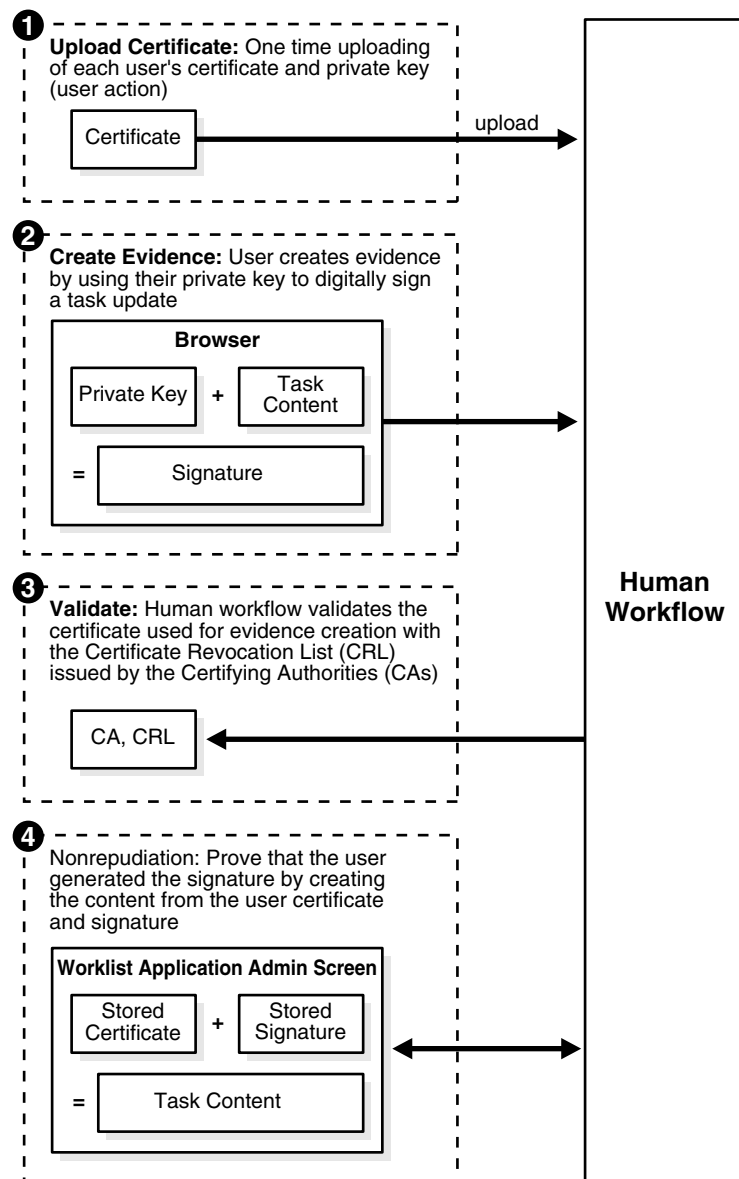
- Is authentic
- Has not been forged by another entity
- Has not been altered
- Cannot be repudiated by the sender

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 Oracle BPM Worklist, when a user approves or rejects the task, the web browser:

- Asks the user to choose the private key to use for signing.
- Generates a digital signature using the private key and task content provided by Oracle BPM Worklist.

Figure 31–2 provides an example.

Figure 31–2 Digital Signature and Certificate**Notes:**

- The certificate refers to a Personal Information Exchange Syntax Standard (PFX) file that includes a certificate and a private key, and is protected by a simple text password. PFX specifies a portable format for storing or transporting a user's private keys, certificates, miscellaneous secrets, and so on.
- The possession of a private key that corresponds to the public key of a certificate is sufficient to sign the data, because the signature is verifiable through the public key in the certificate. However, no attempt is made to correlate the name of a user of a certificate with the person updating it. For example, user `jstein` can sign using the private key of user `cdickens` if `jstein` has that private key.

The following digital signature features are supported:

- PKCS7 signatures based on X.509 certificates
- Browser-based, digitally-signed content without attachments

31.1.10.1 Prerequisites

Prerequisites for using digital signatures and certificates are as follows:

- Users of the Oracle BPM Worklist must have certificates
- The administrator must specify the CAs and corresponding CRL URL whose certificates must be trusted. Users are expected to upload only certificates issued by these CAs. This is done by editing the System MBean Browser in Oracle Enterprise Manager Fusion Middleware Control.
 1. Log in to Oracle Enterprise Manager Fusion Middleware Control.
 2. In the navigator, expand the **SOA** folder.
 3. Right-click **soa-infra**, and select **Administration > System Mbean Browser**.
The System Mbean Browser displays on the right side of the page.
 4. Expand **Application Defined MBeans > oracle.as.soainfra.config > Server: *server_name* > WorkflowConfig > human-workflow**.
 5. Click the **Operations** tab on the right side of the page.
 6. Click **addTrustedCA**.
 7. Provide values for **caName** and **caURL**. You must do this for each certificate in the trust chain. For example, values provided for each invocation may look as shown in [Table 31–10](#).

Table 31–10 *caName and caURL Values*

caName	caURL
CN = Intg, OU =AppServ, O =Oracle, C = US	http://www.oracle.com/Integration%20CRL%20Data.crl
CN = Intg1, OU =AppServ, O =Oracle, C = US	http://www.oracleindia.in.com/Integration%20In.crl
CN = Intg2, OU =AppServ, O =Oracle, C = US	http://www.oracle.us.com/integration.crl

8. Click **Invoke**.

31.1.10.2 Interfaces and Methods

[Table 31–11](#) through [Table 31–14](#) describe the methods in the evidence store service. Package `oracle.bpel.services.security.evidence` corresponds to the evidence service.

Table 31–11 *ITaskEvidenceService Interface*

Method	Description
<code>createEvidence</code>	Creates evidence and stores it in the repository for nonrepudiation.

Table 31–11 (Cont.) ITaskEvidenceService Interface

Method	Description
getEvidence	Gets a list of evidence matching the given criteria. The result also depends on the privileges associated with the user querying the service. If the user has been granted the <code>workflow.admin.evidenceStore</code> permission (points to a location detailing how to grant the permission), all matching evidence is visible. Otherwise, only that evidence created by the user is visible.
uploadCertificate	Uploads certificates to be used later for signature verification. This is a prerequisite for creating evidence using a given certificate. A user can only upload their certificates.
updateEvidence	Updates the CRL verification part of the status. This includes verified time, status, and error messages, if any.
validateEvidenceSignature	Validates the evidence signature. This essentially performs a nonrepudiation check on the evidence. A value of <code>true</code> is returned if the signature is verified. Otherwise, <code>false</code> is returned.

Table 31–12 Evidence Interface

Method	Description
getCertificate	Gets the certificate used to sign this evidence.
getCreateDate	Gets the creation date of the evidence.
getErrorMessage	Gets the error message associated with the CRL validation.
getEvidenceId	Gets the unique identifier associated with the evidence.
getPlainText	Gets the content that was signed as part of this evidence.
getPolicy	Gets the signature policy of the evidence. This is either <code>PASSWORD</code> or <code>CERTIFICATE</code> .
getSignature	Gets the signature of this evidence.
getSignedDate	Gets the date on which the signature was created.
getStatus	Gets the CRL validation status. This can be one of the following: ■ <code>AVAILABLE</code>: The evidence is available for CRL validation. ■ <code>FAILURE</code>: CRL validation failed. ■ <code>SUCCESS</code>: CRL validation succeeded. ■ <code>UNAVAILABLE</code>: The CRL data could not be fetched. ■ <code>WAIT</code>: CRL validation is in progress.
getTaskId	Gets the unique identifier of the task with which this evidence is associated.
getTaskNumber	Gets the task number of the task with which this evidence is associated.
getTaskPriority	Gets the task priority of the task with which this evidence is associated.
getTaskStatus	Gets the task status of the task with which this evidence is associated.
getTaskSubStatus	Gets the task substatus of the task with which this evidence is associated.
getTaskTitle	Gets the title of the task with which this evidence is associated.

Table 31–12 (Cont.) Evidence Interface

Method	Description
<code>getTaskVersion</code>	Gets the version of the task with which this evidence is associated.
<code>getVerifiedDate</code>	Gets the date on which the CRL validation of the certificate used was performed.
<code>getWorkflowType</code>	Gets the workflow type of the task with which this evidence is associated. This is typically <code>BPELWF</code> .

Table 31–13 Certificate Interface

Method	Description
<code>getCA</code>	Gets the certificate issuer's distinguished name (DN).
<code>getCertificate</code>	Gets the certificate object that is abstracted by the interface.
<code>getID</code>	Gets the certificate's serial number.
<code>getIdentityContext</code>	Gets the identity context with which the uploader of this certificate is associated.
<code>getUserName</code>	Gets the user name with whom this certificate is associated.
<code>isValid</code>	Returns <code>true</code> if the certificate is still valid.

Table 31–14 Policy Type and Workflow Type Interface

Method	Description
<code>fromValue</code>	Constructs an object from the string representation.
<code>value</code>	Returns the string representation of this object.

For more information, see the following:

- [Section 27.3.12, "How to Specify a Workflow Digital Signature Policy"](#) for details about specifying digital signatures and digital certificates in the Human Task Editor
- [Chapter 28, "Designing Task Forms for Human Tasks"](#) for details about digitally signing a task action in the Oracle BPM Worklist

31.1.11 Task Instance Attributes

A task is work that must be done by a user. When you create a task, you assign humans to participate in and act upon the task. [Table 31–15](#) describes the task attributes that are commonly used and interpreted by applications.

Table 31–15 Task Attributes

Task Attribute Name	Description
<code>task/applicationContext</code>	The application with which any application roles associated with this task (assignees, owners, and so on) belong.
<code>task/category</code>	An optional category of the task.
<code>task/creator</code>	The name of the creator of this task.
<code>task/dueDate</code>	The due date for the task. This is used on to-do tasks.

Table 31–15 (Cont.) Task Attributes

Task Attribute Name	Description
task/identificationKey	An optional, custom, unique identifier for the task. This can be set as an additional unique identifier to the standard task ID and task number. This key can retrieve a task based on business object identifiers for which the task is created.
task/identityContext	The identity realm under which the users and groups are seeded. In a single realm environment, this defaults to the default realm.
task/ownerGroup	The group (if any) that owns this task instance. Task owners can be application roles, users, or groups. If the owner of the task is a group, this field is set.
task/ownerRole	The application role (if any) that owns this task instance. Task owners can be application roles, users, or groups. If the owner of the task is an application role, this field is set.
task/ownerUser	The user (if any) that owns this task instance. Task owners can be application roles, users, or groups. If the owner of the task is a user, this field is set.
task/payload	The task payload that is captured as XML.
task/percentageComplete	The percentage of the task completed. This is used on to-do tasks.
task/priority	An integer number that defines the priority of this task. A lower number indicates a higher priority. The numbers 1 to 5 are typically used.
task/startDate	The start date for the task. This is used on to-do tasks.
task/subCategory	An optional subcategory of the task.
task/taskDefinitionId	The task definition ID that binds the task to the task metadata. At task initiation time, this can be either the compositeDN/componentName string or the targetNamespace in the .task file. If the later is used, the active version matching the targetNamespace is used.
task/taskDisplayUrl	The URL to use to display the details for this task.
task/title	The title of the task.

Table 31–16 lists the attributes that capture process metadata information.

Table 31–16 Attributes Capturing Process Metadata Information

Attribute	Description
task/processInfo/domain	The domain to which the composite that contains the task component that defines this task instance belongs.
task/sca/applicationName	The application that is deployed.
task/sca/componentName	The name of the task component that defines this task instance.
task/sca/compositeDN	A unique name for the particular deployment of the composite that contains the task component that defines this task instance.
task/sca/compositeInstanceId	The composite instance ID.
task/sca/compositeName	The name of the composite that contains the task component that defines this task instance.

Table 31–16 (Cont.) Attributes Capturing Process Metadata Information

Attribute	Description
task/sca/compositeVersion	The version of the composite that contains the task component that defines this task instance.

Table 31–17 lists the attachment-related attributes.

Table 31–17 Attachment-related attributes

Attribute	Description
task/attachment/content	The attachment content.
task/attachment/mimeType	The Multipurpose Internet Mail Extension (MIME) type of the attachment.
task/attachment/name	The name of the attachment.
task/attachment/updatedBy	The user who updated the attachment.
task/attachment/updatedAt	The date on which the attachment was updated.
task/attachment/URI	The URI if the attachment is URI-based.

Table 31–18 lists the comment-related attributes.

Table 31–18 Comment-related Attributes

Attribute	Description
task/userComment/comment	The user comment.
task/userComment/addedBy	The user who added the comment.
task/userComment/addedDate	The date on which the comment was added.

Table 31–19 lists the attributes manipulated by the workflow services system.

Table 31–19 Attributes Manipulated by the Workflow Services System

Attribute	Description
task/systemAttributes/acquiredBy	If a task is assigned to a group, application role, or to multiple users, and then claimed by a user, this field is set to the name of the user who claimed the task.
task/systemAttributes/approvers	The IDs of users who performed custom actions on the task.
task/systemAttributes/assignedDate	The date that this task was assigned.
task/systemAttributes/assignees	The current task assignees (can be users, groups, or application roles).
task/systemAttributes/createdDate	The date the task instance was created.
task/systemAttributes/customActions	The custom actions that can be performed on the task.
task/systemAttributes/endDate	The end date for the task. This is used on to-do tasks.

Table 31–19 (Cont.) Attributes Manipulated by the Workflow Services System

Attribute	Description
task/systemAttributes/expirationDate	The date on which the task instance expires.
task/systemAttributes/fromUser	The user who previously acted on the task.
task/systemAttributes/hasSubTasks	If true, there are subtasks.
task/systemAttributes/isGroup	If true, the task is assigned to a group.
task/systemAttributes/originalAssigneeUser	If a user delegates a task to another user, this field is populated with the name of the user who delegated the task.
task/systemAttributes/outcome	The outcome of the task (for example, approved or rejected). This is only set on completed task instances.
task/systemAttributes/parentTaskId	This is only set on reinitiated tasks (the task ID of the previous task that is being reinitiated).
task/systemAttributes/parentTaskVersion	This only set on a subtask. This refers to the version of the parent task.
task/systemAttributes/participantName	The logical name of the participant as modeled from Oracle JDeveloper.
task/systemAttributes/reviewers	The reviewers of the task. This can be a user, group, or application role.
task/systemAttributes/rootTaskId	The ID of the root task. This is the same as the task ID for the root task.
task/systemAttributes/stage	The stage name that is being executed.
task/systemAttributes/state	The current state of the task instance.
task/systemAttributes/substate	The current substate of the task.
task/systemAttributes/subTaskGroupId	A unique ID that is set on a subtask. This same ID is set on the parent task's taskGroupId. This is required to identify which subtasks were created at which time.
task/systemAttributes/systemActions	The system actions (such as reassign, escalate, and so on) that can be performed on a task.
task/systemAttributes/taskDefinitionName	The name of the task component that defines this task instance.
task/systemAttributes/taskGroupId	The ID of the immediate parent task. This only sets a subtask.
task/systemAttributes/taskGroupId	A unique ID that is set on the parent task. This same ID is set on the subtask's subTaskGroupId. This is required to identify which subtasks were created at which time.
task/systemAttributes/taskId	The unique ID of the task.
task/systemAttributes/taskNamespace	A namespace that uniquely defines all versions of the task component that defines this task instance. Different versions of the same task component can have the same namespace, but no two task components can have the same namespace.
task/systemAttributes/taskNumber	An integer number that uniquely identifies this task instance.

Table 31–19 (Cont.) Attributes Manipulated by the Workflow Services System

Attribute	Description
task/systemAttributes/updatedBy	The user who last updated the task.
task/systemAttributes/updatedAt	The date this instance was last updated.
task/systemAttributes/version	The version of the task.
task/systemAttributes/versionReason	The reason the version was created.
task/systemAttributes/workflowPattern	The pattern that is being executed (for example, parallel, serial, FYI, or single).

Table 31–20 lists the mapped attributes.

Table 31–20 Mapped Attributes

Attribute	Description
task/systemMessageAttributes/*	The mapped attributes.

31.2 Notifications from Human Workflow

Notifications are sent to alert users of changes to the state of a task. Notifications can be sent through any of the following channels: email, telephone voice message, instant messaging (IM), or short message service (SMS). Notifications can be sent from a human task in a BPEL process or directly from a BPEL process.

In releases before 11g, email notifications were sent through the human workflow email notification layer. Voice and SMS notifications were sent through Oracle's hosted notification service. IM notifications were not supported.

Starting with release 11g, the human workflow email notification layer works with Oracle User Messaging Service to alert users to changes in the state of a task. The Oracle User Messaging Service exposes operations that can be invoked from the BPEL process or human task to send notifications through email, voice, IM, or SMS channels.

The Oracle User Messaging Service supports features such as:

- Sending and receiving messages and statuses
- Sending notifications to a specific address on a particular channel
- Sending notifications to a set of failover addresses

On application servers other than Oracle Fusion Middleware, the human workflow email notification layer can be used for email notifications.

For more information about configuring the Oracle User Messaging Service, see the following:

- [Chapter 16, "Using the Notification Service"](#)
- [Part XI, "Using Oracle User Messaging Service"](#)
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for instructions on configuring notification service delivery channels in Oracle Enterprise Manager Fusion Middleware Control

31.2.1 Contents of Notification

Each email notification can contain the following parts:

- The notification message
- The HTML content from Oracle BPM Worklist:

This is a read-only view of Oracle BPM Worklist on the task. For information on how you can configure email notifications to include the content from Oracle BPM Worklist, see [Section 28.7, "Creating an Email Notification."](#)
- Task attachments:

For notifications that include task attachments.
- Actionable links

Notifications through SMS, IM, and voice contain *only* the notification message.

The notification message is an XPath expression that can contain static text and dynamic values. In creating the messages, only the task BPEL variable is available for dynamic values. This restriction is because the messages are evaluated outside the context of the BPEL process. The payload in the task variable is also strongly typed to contain the type of the payload for XPath tree browsing. The XPath extension function `hwf:getNotificationProperty(propertyName)` is available to get properties for a particular notification. The function evaluates to corresponding values for each notification. The `propertyName` can be one of the following values:

- `recipient`

The recipient of the notification
- `recipientDisplay`

The display name of the recipient
- `taskAssignees`

The task assignees
- `taskAssigneesDisplay`

The display names of the task assignees
- `locale`

The locale of the recipient
- `taskId`

The ID of the task for which the notification is meant
- `taskNumber`

The number of the task for which the notification is meant
- `appLink`

The HTML link to the Oracle BPM Worklist task details page

[Example 31-4](#) demonstrates the use of `hwf:getNotificationProperty` and `hwf:getTaskResourceBundle`:

Example 31-4 Use of `hwf:getNotificationProperty` and `hwf:getTaskResourceBundle`

```
concat('Dear ', hwf:getNotificationProperty('recipientDisplay'), ' Task ',  
/task:task/task:systemAttributes/task:taskNumber, ' is assigned to you. ',  
hwf:getTaskResourceBundleString(/task:task/task:systemAttributes/task:taskId,
```



```
'CONGRATULATIONS', hwf:getNotificationProperty('locale')))
```

This results in a message similar to the following:

Dear Cooper, James Task 1111 is assigned to you. Congratulations

31.2.2 Error Message Support

The human workflow email notification layer is automatically configured to warn an administrator about error occurrences in which intervention is required. Error notifications and error response messages are persisted.

You can view messages in Oracle Enterprise Manager Fusion Middleware Control.

For more information about viewing messages, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

31.2.3 Reliability Support

The human workflow email notification layer works with Oracle User Messaging Service to provide the following reliability support:

- Messages are not lost:
 - If the human workflow email notification layer fails after acknowledging receipt of a message from the human workflow.
 - If the human workflow email notification layer and Oracle User Messaging Service both fail before the Oracle User Messaging Service acknowledges receipt of a message from the human workflow.
 - If the Oracle User Messaging Service is down. Message delivery is retried until successful.
 - If a notification channel is down.
- Notifications that cannot be delivered are retried three times and the address is marked as invalid. The address is also added to the bad address list. If needed, you can manually remove these addresses from the bad address list in Oracle Enterprise Manager Fusion Middleware Control. Outgoing notifications are not resent until the address is corrected. To guard against any incorrect identification, the address is marked as invalid only for about an hour. No new notifications are sent in this time.
- Incoming notification responses from an address that has been identified as a spam source are ignored.
- Incoming notification messages are persisted.
- Incoming notification responses that indicate notification delivery failure (for example, an unknown host mail) are not ignored. Instead, corrective actions are automatically taken (for example, the bad address list is updated).
- Incoming notification responses can be configured to send acknowledgements indicating notification status to the sender.
- Validation of incoming notification responses is performed by correlating the incoming notification message with the outgoing notification message.

For more information about notifications, see the following:

- [Chapter 16, "Using the Notification Service"](#)

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

31.2.4 Management of Oracle Human Workflow Notification Service

An administrator can perform the following management tasks from Oracle Enterprise Manager Fusion Middleware Control:

- View failed notifications and erroneous incoming notification responses and take corrective actions.
- Perform corrective actions such as delete, resend, and edit on outgoing notifications and addresses.
- View bad emails and block email addresses for incoming notification responses.
- Manage the bad email address list.
- Access runtime data of failed notifications. You can purge this data when it is no longer needed.

For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

31.2.5 How to Configure the Notification Channel Preferences

To configure the notification channel preferences:

1. In Oracle JDeveloper, configure the notification service for email and other channels. See [Chapter 16, "Using the Notification Service"](#) for details.
2. Open the Human Task Editor in Oracle JDeveloper.

The notifications for a task can be configured during the creation of a task in the Human Task Editor. Notifications can be sent to different types of participants for different actions.

The actions for which a task notification can be sent are described in [Section 27.3.10.1, "Notifying Recipients of Changes to Task Status."](#)

Notifications can be sent to users involved in the task in various capacities. These users are described in [Section 27.3.10.1, "Notifying Recipients of Changes to Task Status."](#)

When the task is assigned to a group, each user in the group is sent a notification if no notification endpoint is available for the group.

For more information, see the following:

- [Chapter 16, "Using the Notification Service"](#)
 - [Section 27.3.10, "How to Specify Participant Notification Preferences"](#) to configure task notifications in the Human Task Editor
 - *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for details about configuring the notification channel
3. From the messaging server pages of Oracle Enterprise Manager Fusion Middleware Control, configure the appropriate channel (for example, email). See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for details.

4. From the Workflow Notification Properties pages of Oracle Enterprise Manager Fusion Middleware Control, configure the notification mode parameter for the notification service to either all channels or email.

By default, this value is set to **NONE**, meaning that no notifications are sent. The possible values are:

- **ALL**

The email, IM, SMS, and voice channels are configured and notification is sent through any channel.

- **EMAIL**

Only the email channel is configured for sending notification messages.

- **NONE**

No channel is configured for sending notification messages. This is the default setting.

31.2.6 How to Configure Notification Messages in Different Languages

A notification consists of four types of data generated from multiples sources and internationalized differently. However, for all internationalized notifications, the locale is obtained from the `BPMUser` object of the identity service.

- **Prepackaged strings** (action links, comments, Oracle BPM Worklist, and so on)

These strings are internationalized in the product as part of the following package:

```
oracle.bpel.services.workflow.resource
```

The user's locale is used to get the appropriate message.

- **Task details attachment**

The user's locale is used to retrieve the task detail HTML content.

- **Task outcome strings** (approve, reject, and so on)

The resource bundle for outcomes is specified when the task definition is modeled in the **Advanced Settings** section of the Human Task Editor. The key to each of the outcomes in the resource bundle is the outcome name itself.

- **Notification message**

To configure notification messages in different languages:

1. Use one of the following methods to internationalize messages in the notification content:

- a. To use values from the resource bundle specified during the task definition, use the following XPath extension function:

```
hwf:getTaskResourceBundleString(taskId, key, locale?)
```

This function returns the internationalized string from the resource bundle specified in the task definition.

The locale of the notification recipient can be retrieved with the following function:

```
hwf:getNotificationProperty('locale')
```

The task ID corresponding to a notification can be retrieved with the following function:

```
hwf:getNotificationProperty('taskId')
```

- b.** If a different resource bundle is used, then use the following XPath extension to retrieve localized messages:

```
orcl:get-localized-string()
```

For more information, see [Section 27.3.8.2, "Specifying Multilingual Settings."](#)

31.2.7 How to Send Actionable Messages

There are several methods for sending actionable messages. This section provides an overview of procedures.

Note: If digital signatures are enabled for a task, actionable emails are not sent during runtime. This is the case even if actionable emails are enabled during design time.

31.2.7.1 How to Send Actionable Emails for Human Tasks

Task actions can be performed through email if the task is set up to enable actionable email (the same actions can also be performed from Oracle BPM Worklist). An actionable email account is the account in which task action-related emails are received and processed.

To send actionable emails for human tasks:

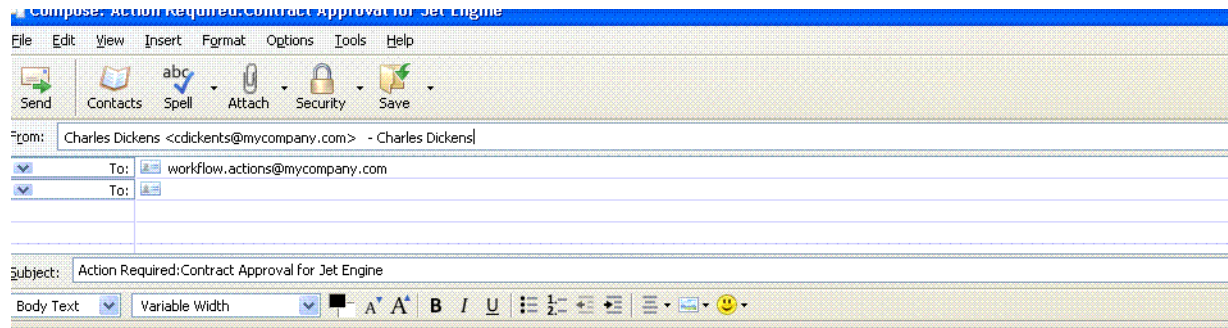
1. In the **Advanced** tab of the **Notification** section of the Human Task Editor, select **Make notification actionable** to make email notifications actionable. This action enables you to perform task actions through email.

If a notification is actionable, the email contains links for each of the custom outcomes.

2. To send task attachments with the notification message, select **Send task attachments with email notifications**.

When an actionable email arrives, perform the following tasks.

3. Click the **Approve** link to invoke a new email window with approval data. [Figure 31–3](#) provides details.

Figure 31–3 Actionable Notifications

Add comments by editing the text between the brackets below.

Comments: [This contract has been approved based on attached information.]

You can also add attachments to the task by attaching them to this email.

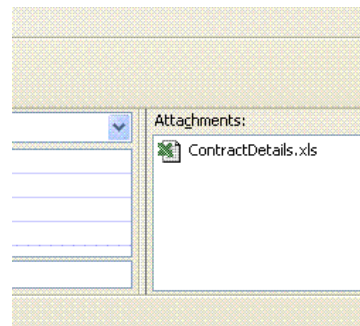
-----Do not edit below this line-----

Approve : [[NID]] : 8Scnzw12KyJYLkTs+bJJUaxW5IC96bzHKvEXkXJø8Ogk9opI1QQuDaNI7vq0Jq3D7DqYCRCS1bø1GMXqHCrUjUYAQQvds [[NID]]

4. Add comments in the comments section of the approval mail. For example:

This contract has been approved based on the attached information.

5. Add attachments as needed, as shown in [Figure 31–4](#).

Figure 31–4 Attachment to an Actionable Email

6. Do not change anything in the subject or the body in this email. If you change the content with the NID substrings, the email is not processed.
7. Click **Send**.
8. Set properties such as incoming server, outgoing mail server, outgoing username and password, and others from the Oracle User Messaging Service section of Oracle Enterprise Manager Fusion Middleware Control.
9. In the Workflow Notification Properties page of Oracle Enterprise Manager Fusion Middleware Control, set the notification mode to **ALL** or **EMAIL**.
10. In the Workflow Task Service Properties page of Oracle Enterprise Manager Fusion Middleware Control, set the actionable email account name.

For more information about the Oracle User Messaging Service section, Workflow Notification Properties page, and Workflow Task Service Properties page of Oracle Enterprise Manager Fusion Middleware Control, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

31.2.8 How to Send Inbound and Outbound Attachments

If the include attachments flag is checked; only email is sent. The emails include all the task attachments as email attachments.

To send inbound and outbound attachments:

1. Select **Send task attachments with email notifications** in the **Advanced** tab of the **Notification** section of the Human Task Editor.

In the actionable email reply, the user can add attachments in the email. These attachments are added as task attachments.

For more information, see [Section 27.3.10.7, "Making Email Messages Actionable."](#)

31.2.9 How to Send Inbound Comments

To send inbound comments:

1. Add comments in the actionable email reply between `Comments [[` and `]]`, as shown in [Figure 31–3](#). Those contents are added as task comments. For example, `Comments [[looks good]]`.

31.2.10 How to Send Secure Notifications

To send secure notifications:

1. Select **Make notifications secure (exclude details)** in the **Advanced** tab of the **Notification** section of the Human Task Editor. This action enables a default notification message to be used. In this case, the notification message does not include the content of the task. Also, this notification is not actionable. The default notification message includes a link to the task in Oracle BPM Worklist. You must log in to see task details.

For more information, see [Section 27.3.10.5, "Securing Notifications to Exclude Details."](#)

31.2.11 How to Set Channels Used for Notifications

To set channels used for notifications:

1. Set up preferred notification channels by using the preferences user interface in Oracle BPM Worklist. The channel is dynamically determined by querying the user preference store before sending the notification. If the user preference is not specified, then the email channel is used.

For more information about the Oracle Delegated Administration Service, see *Oracle Fusion Middleware Guide to Delegated Administration for Oracle Identity Management*.

31.2.12 How to Send Reminders

Tasks can be configured to send reminders, which can be based on the time the task was assigned to a user or the expiration time of a task. The number of reminders and the interval between the reminders can also be configured. The message used for reminders is the message that is meant for ASSIGNEES when the task is marked as ASSIGNED.

To send reminders:

1. Set reminders in the **Advanced** tab of the **Notification** section of the Human Task Editor. Reminder configuration involves the following parameters:
 - **Recurrence:**
Specifies the number of times reminders are sent. The possible values for recurrence are EVERY, NEVER, 0, 1, 2 ..., 10.
 - **RelativeDate:**
Specifies if the reminder duration is computed relative to the assigned date or to the expiration date of the task. The possible values for the `relativeDate` are ASSIGNED, EXPIRATION, and BEFORE DUE DATE. The final value appears in Oracle JDeveloper if you modify the escalation and expiration policy of the task to use the option **Action Requested Before** (known as **Use Due Date** in previous releases).
 - **Duration:**
Specifies the duration from the `relativeDate` and the first reminder and each reminder since then. The data type of duration is `xsd:duration`, whose format is defined by ISO 8601 under the form `PnYnMnDTnHnMnS`. The capital letters are delimiters and can be omitted when the corresponding member is not used. Examples include PT1004199059S, PT130S, PT2M10S, P1DT2S, -P1Y, or P1Y2M3DT5H20M30.123S.

The following examples illustrate when reminders are sent:

- If the `relativeDate` is ASSIGNED, the recurrence is EVERY, the reminder duration is PT1D, and the task is assigned at 3/24/2005 10:00 AM, then reminders are sent at 3/25/2005 10:00 AM, 3/26/2005 10:00 AM, 3/27/2005 10:00 AM, and so on until the user acts on the task.
- If the `relativeDate` is EXPIRATION, the recurrence is 2, the reminder duration is PT1D, and the task expires at 3/26/2005 10:00 AM, then reminders are sent at 3/24/2005 10:00 AM and 3/25/2005 10:00 AM if the task was assigned before 3/24/2005 10:00 AM.
- If the `relativeDate` is EXPIRATION, the recurrence is 2, the reminder duration is PT1D, the task expires at 3/26/2005 10:00 AM, and the task was assigned at 3/24/2005 3:00 PM, then only one reminder is sent at 3/25/2005 10:00 AM.

For more information, see [Section 27.3.10.3, "Setting Up Reminders."](#)

31.2.13 How to Set Automatic Replies to Unprocessed Messages

The human workflow notification service sends you an automatic reply message when it cannot process an incoming message (due to system error, exception error, user error, and so on). You can modify the text for these messages in the global resource bundle.

[Example 31-5](#) shows the `WorkflowLabels.properties` file. For more information, see [Section 31.5.2, "Global Resource Bundle – WorkflowLabels.properties."](#)

Example 31-5 WorkflowLabels.properties

```
# String to be prefixed to all auto reply messages
AUTO_REPLY_PREFIX_MESSAGE=Oracle Human Workflow Service
# String to be suffixed to all auto reply messages
AUTO_REPLY_SUFFIX_MESSAGE=This message was automatically generated by Human \
Workflow Mailer. Do not reply to this mail.
```

```
# Message indicating closed status of a notified task
TaskClosed=You earlier received the notification shown below. That notification \
          is now closed, and no longer requires your response. You may \
          simply delete it along with this message.

# Message indicating that notification was "replied" to instead of "responded" by
# using the response link.
EmailRepliedNotification=The message you sent appeared to be a reply to a \
          notification. To respond to a notification, use the \
          response link that was included with your notification.

#
EmailUnSolicited= The message you sent did not appear to be in response to a \
          notification. If you are responding to a notification      \
          Use the response link that was included with your notification.

EmailUnknownContent= The message you sent did not appear to be in response to a \
          notification. If you are responding to a notification,      \
          Use the response link that was included with your notification.

ResponseNotProcessed=Your response to notification could not be processed. \
          Log in to worklist application for more details.

ResponseProcessed=Your response to notification was successfully processed.
```

31.2.14 How to Create Custom Notification Headers

Some task participants may have access to multiple notification channels. You can use custom notification headers to enable this type of participant to specify a single channel as the preferred channel on which to receive notifications.

To create custom notification headers:

1. In the **Notification header attributes** section of the **Advanced** tab of the **Notification** section of the Human Task Editor, create custom notification headers that specify the preferred notification channel to use (such as voice, SMS, and so on). The human workflow email notification layer provides these header values to the rule-based notification service of the Oracle User Messaging Service for use.

For example, set the **Name** field to `deliveryType` and the **Value** field to `SMS`.

Note that the rule-based notification service is *only* used to identify the preferred notification channel to use. The address for the preferred channel is obtained from Oracle Identity Management. The notification message is created from the information provided by both services.

For more information, see the following:

- [Section 27.3.10.8, "Sending Task Attachments with Email Notifications"](#)
- [Chapter 64, "User Messaging Preferences"](#)

31.3 Assignment Service Configuration

This section describes how to configure the assignment service with dynamic assignment functions.

31.3.1 Dynamic Assignment and Task Escalation Functions

When tasks are assigned to a group, users in the group must typically claim a task to act on it. However, you can also automatically send work to users in the group by using various dispatching mechanisms. Automatic task dispatching is done through dynamic assignment functions. Dynamic assignment functions select a particular user or group from either a group, or from a list of users or groups. Several functions are automatically provided. However, you can also create your own functions and register them with the workflow service. [Table 31–21](#) describes the three dynamic assignment functions.

Table 31–21 *Dynamic Assignment Functions*

Function	Type	Description
LEAST_BUSY	Dynamic assignment	Picks the user or group with the least number of tasks currently assigned to it.
MANAGERS_MANAGER	Task escalation	Picks the manager's manager.
MOST_PRODUCTIVE	Dynamic assignment	Picks the user or group that has completed the most tasks over a certain time period (by default, the last seven days).
ROUND_ROBIN	Dynamic assignment	Picks each user or group in turn.

These functions all check a user's vacation status. A user that is currently unavailable is not automatically assigned tasks.

These dynamic assignment functions can be called using the custom XPath functions in a BPEL process or task definition:

- `wfDynamicUserAssign`
- `wfDynamicGroupAssign`

These XPath functions must be called with at least two, and optionally more, parameters:

- The name of the dynamic assignment function being called.
- The name of the group on which to execute the function (or a list of users or groups).
- (Optional) The identity realm to which the user or group belongs (the default value is the default identity realm).
- Additional optional parameters specific to the dynamic assignment function. In the case of the `MOST_PRODUCTIVE` assignment function, this is the length of time (in days) to calculate a user's productivity. The two other functions do not use additional parameters.

In addition, human workflow rules created for a group can use dynamic assignment functions to select a member of that group for reassignment of a task.

In addition to these functions, a dynamic assignment framework is provided that enables you to implement and configure your own dynamic assignment functions.

31.3.1.1 How to Implement a Dynamic Assignment Function

Follow these procedures to implement your own dynamic assignment function.

To implement dynamic assignment functions:

1. Write a Java class that implements one or both of the following interfaces:

```
oracle.bpel.services.workflow.assignment.dynamic.  
IDynamicUserAssignmentFunction  
oracle.bpel.services.workflow.assignment.dynamic.  
IDynamicGroupAssignmentFunction
```

2. If your dynamic assignment function selects users, implement the first interface. If it selects groups, implement the second interface. If it allows the selection of both users and groups, implement both interfaces.

The two interfaces above both extend the interface

```
oracle.bpel.services.workflow.assignment.dynamic.IDynamicAssignmentFunction.
```

Your Java class should also implement the methods in that interface. These interfaces are shown in the Javadoc.

The dynamic assignment framework also provides the utility class

```
oracle.bpel.services.workflow.assignment.dynamic.DynamicAssignmentUtils.
```

This class provides many methods that are useful when implementing dynamic assignment functions.

For information about the Javadoc for dynamic assignment interfaces and utilities, see *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*.

31.3.1.2 How to Configure Dynamic Assignment Functions

Dynamic assignment functions are configured along with other human workflow configuration parameters in Oracle Enterprise Manager Fusion Middleware Control.

Each dynamic assignment has two mandatory parameters in this file, in the form of a `<function>` tag.

The function tag must contain two attributes:

- `name:`
The name of the function

- `classpath:`
The fully qualified class name of the class that implements the function.

In addition, each function can optionally have any number of properties. These properties are simple name-value pairs that are passed as initialization parameters to the function.

The property values specified in these tags are passed as a map (indexed by the value of the name attributes) to the `setInitParameters` method of the dynamic assignment functions.

Two of the functions have initialization parameters. These are:

- `ROUND_ROBIN`
The parameter `MAX_MAP_SIZE` specifies the maximum number of sets of users or groups for which the function can maintain `ROUND_ROBIN` counts. The dynamic assignment function holds a list of users and groups in memory for each group (or

list of users and groups) on which it is asked to execute the `ROUND_ROBIN` function.

- **MOST_PRODUCTIVE**

The parameter `DEAFULT_TIME_PERIOD` specified the length of time (in days) over which to calculate the user's productivity. This value can be overridden when calling the `MOST_PRODUCTIVE` dynamic assignment function. Use an XPath function by specifying an alternative value as the third parameter in the XPath function call.

For more information about configuring the dynamic assignment functions from Oracle Enterprise Manager Fusion Middleware Control, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

31.3.1.3 How to Configure Display Names for Dynamic Assignment Functions

The runtime config service provides methods for returning a list of available user and group dynamic assignment functions. These functions return both the name of the function, and a user-displayable label for the function. The functions support localization of the display name, so that it displays in the appropriate language for the context user. These functions are used by Oracle BPM Worklist to show a list of available dynamic assignment functions. This applies exclusively to dynamic assignment functions. Display names for task escalation functions are not supported.

To configure display names for dynamic assignment functions:

1. Specify display names (and appropriate translations) for your dynamic assignment functions by adding entries to the resource property file `WorkflowLabels.properties`, and associated resource property files in other languages. This file exists in the class path identified in the workflow configuration parameter `workflowCustomizationsClasspathURL`.

Entries for dynamic assignment functions must be of the form:

```
DYN_ASSIGN_FN.[function name]=Function Display Name
```

For instance, the entry for the `ROUND_ROBIN` function is:

```
DYN_ASSIGN_FN.ROUND_ROBIN = Round Robin
```

Note that adding entries to these files for dynamic assignment functions is optional. If no entry is present in the file, then the name of the function (for example, `ROUND_ROBIN`) is used instead.

For more information about the `WorkflowLabels.properties` file, see the `workflow-110-workflowCustomizations` sample available at the following URL:

<https://soasamples.samplecode.oracle.com/>

31.3.1.4 How to Implement a Task Escalation Function

Task escalation functions are very similar to dynamic assignment functions, but perform a different function (determining to whom a task is assigned when it is escalated). Custom implementations must implement a different interface (`IDynamicTaskEscaltionFunction`).

31.3.2 Dynamically Assigning Task Participants with the Assignment Service

Human workflow participants are specified declaratively in a routing slip. The routing slip guides the human workflow by specifying the participants and how they

participate in the human workflow (for example, management chain hierarchy, serial 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, to do more sophisticated routing using custom logic, you implement a custom assignment service to do routing. To support a dynamic assignment, an assignment service is used. The assignment service is responsible for determining the task assignees. You can also implement your own assignment service and plug in that implementation for use with a particular human workflow.

The assignment service determines the following task assignment details in a human workflow:

- The assignment when the task is initiated.
- The assignment when the task is reinitiated.
- The assignment when a user updates the task outcome. When the task outcome is updated, the task can either be routed to other users or completed.
- The assignees from whom information for the task can be requested.
- If the task supports reapproval from Oracle BPM Worklist, a user can request information for reapproval.
- The users who reapprove the task if reapproval is supported.

The human workflow service identifies and invokes the assignment service for a particular task to determine the task assignment.

For example, a simple assignment service iteration is as follows:

1. A client initiates an expense approval task whose routing is determined by the assignment service.
2. The assignment service determines that the task assignee is `jcooper`.
3. When `jcooper` approves the task, the assignment service assigns the task to `jstein`. The assignment service also specifies that a notification must be sent to the creator of the task, `jlonson`.
4. `jstein` approves the task and the assignment service indicates that there are no more users to whom to assign the task.

31.3.2.1 How to Implement an Assignment Service

To implement an assignment service:

1. Implement the assignment service with the `IAssignmentService` interface. The human workflow service passes the following information to the assignment service to determine the task assignment:
 - Task document
The task document that is executed by the human workflow. The task document contains the payload and other task information like current state, and so on.
 - Map of properties

When an assignment service is specified, a list of properties can also be specified to correlate callbacks with back-end services that determine the task assignees.

- Task history

The task history is a list of chronologically-ordered task documents to trace the history of the task. The task documents in this list contain a subset of attributes in the actual task (such as `state`, `updatedBy`, `outcome`, `updatedAt`, and so on).

31.3.2.2 Example of Assignment Service Implementation

Notes:

- The assignment service class cannot be stateful because every time human workflow services must call the assignment service, it creates a new instance.
 - The `getAssigneesToRequestForInformation` method can be called multiple times because one of the criteria to show the request-for-information action is that there are users to request information. Therefore, this method is called every time the human workflow service tries to determine the permitted actions for a task.
-
-

You can implement your own assignment service plug-in that the human workflow service invokes during human workflow execution.

[Example 31-6](#) provides a sample `IAssignmentService` implementation named `TestAssignmentService.java`.

Example 31-6 Sample `IAssignmentService` Implementation

```
/* $Header: TestAssignmentService.java 24-may-2006.18:26:16 Exp $ */
/* Copyright (c) 2004, 2006, Oracle. All rights reserved. */
/*
DESCRIPTION
Interface IAssignmentService defines the callbacks an assignment
service implements. The implementation of the IAssignmentService
is called by the workflow service
PRIVATE CLASSES
<list of private classes defined - with one-line descriptions>
NOTES
<other useful comments, qualifications, etc.>
MODIFIED      (MM/DD/YY)

*/
/**
 * @version $Header: IAssignmentService.java 29-jun-2004.21:10:35 Exp
 *
 *
 */
package oracle.bpel.services.workflow.test.workflow;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import oracle.bpel.services.workflow.metadata.routing.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;
    }
}

```

31.3.2.3 How to Deploy a Custom Assignment Service

To deploy a custom assignment service:

1. Use one of the following methods to make an assignment service implementation class and its related classes available in the class path of Oracle BPEL Process Manager:
 - Load your classes in `SCA-INF/classes` directly or in `SCA-INF/lib` as a JAR.
 - Place the class files for your custom function in a directory tree or JAR file. Then, update the `workflowCustomClasspathURL` configuration parameter to point to the JAR or root directory in which your classes are located. As this is a URL, it is possible to host the class files on a web server, and make them accessible to multiple Oracle WebLogic Servers through HTTP. It is even possible to deploy the files into the metadata repository (MDS), and use an ORAMDS URL to point to the appropriate location. This approach is described in detail in sample workflow-110-workflowCustomizations. To download this sample, visit the following URL:

<https://soasamples.samplecode.oracle.com/>

Notes:

- 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.
-

31.3.3 Custom Escalation Function

The custom escalation function enables you to integrate a custom rule in a human workflow.

To implement a custom escalation function:

1. Create a custom task escalation function and register this with the human workflow service that uses that function in task definitions.
2. Use the Human Task Editor to integrate the rule in a human workflow.

For more information, see [Section 27.3.9.6, "Specifying Escalation Rules."](#)

31.4 Class Loading for Callbacks and Resource Bundles

You can load classes for the following callbacks and resource bundles directly from the SOA project instead of having to load classes in the `oracle.soainfra.common` shared library and restarting Oracle WebLogic Server:

- `IAssignmentService`
- `IRestrictedAssignmentService`
- `IRoutingSlipCallback`
- `IPercentageCompletionCallback`
- `INotificationCallback`
- Project level resource bundles

The callback classes can be in the following locations:

- JARs in the `SCA-INF/lib` directory of the project
- Classes in the `SCA-INF/classes` directory of the project

Additionally, to support backward compatibility, the project level resource bundles can also be in the same directory as the `.task` file.

31.5 Resource Bundles in Workflow Services

This section describes the resource bundles used in human workflow services and how they can be customized to provide alternative resource strings.

The human workflow service APIs and Oracle BPM Worklist use the locale set in the `IWorkflowContext` object to access the APIs. This is the locale of the user in the user directory configured with the identity service. If no locale is specified for the user, then the default locale for the Java EE server is used instead.

It is possible for API clients to override this locale by setting a new value in the `IWorkflowContext` object. Oracle BPM Worklist provides a user preference option that allows users to use their browser's locale, rather than the locale set in their user directory.

31.5.1 Task Resource Bundles

Each human workflow component can be associated with a resource bundle. The bundle defines the resource strings to use as display names for the task outcomes. The resource strings are returned by the `TaskMetadataService` method `getTaskDefinitionOutcomes`, and are displayed in Oracle BPM Worklist and the task flow task details application.

In addition, you can use the human workflow XPath function `getTaskResourceBundle` string to look up resource strings for the task's resource bundle. For example, this XPath function can be used as part of the XPath expression used to construct notification messages for the task.

A human workflow component is associated with a resource bundle by setting the **Resource Name** and **Resource Location** fields of the Resource Details dialog in the **Presentation** section of the Human Task Editor. Note that the value for the **Resource Location** field is a URL, and the resource bundle can be contained within a JAR file pointed to by the URL. It is possible to share the same resource bundle between multiple human workflow components by using a common location for the resource bundle.

If no resource bundle is specified for the human workflow component, the resource string is looked up in the global resource bundle. (See [Section 31.5.2, "Global Resource Bundle – WorkflowLabels.properties."](#)) Commonly-used task outcomes can be defined in the global resource bundle, alleviating the need to define a resource bundle for individual human workflow components.

If no resource string can be located for a particular outcome, then the outcome name is used as the display value in all locales.

31.5.2 Global Resource Bundle – WorkflowLabels.properties

The following global resource bundle is used by human workflow service APIs to look up resource strings:

```
oracle.bpel.services.workflow.resource.WorkflowLabels.properties
```

You can customize this bundle to provide alternatives for existing display strings or to add additional strings (for example, for mapped attribute labels, standard views, or custom dynamic assignment functions).

The global resource bundle provides resource strings for the following:

- Task attributes:

Labels for the various task attributes displayed in Oracle BPM Worklist (or other clients). Resource string values are returned from the following TaskMetadataService methods:

- `getTaskAttributes`
- `getTaskAttributesForTaskDefinition`
- `getTaskAttributesForTaskDefinitions`

- Mapped attribute labels:

Mapped attribute labels created through the runtime config service. These strings are used in Oracle BPM Worklist when displaying mapped attributes. Resource string values are returned from the TaskMetadataService methods:

- `getTaskAttributesForTaskDefinition`
- `getTaskAttributesForTaskDefinitions`

If translated resource strings are required for mapped attribute mappings, then customize the `WorkflowLabels.properties` bundle to include the appropriate strings.

- Task outcomes:

Default resource strings for common task outcomes. These can be overridden by the task resource bundle, as described above. The resource strings are returned by the TaskMetadataService method `getTaskDefinitionOutcomes`, if no task-specific resource bundle has been specified. As shipped, the global resource bundle contains resource strings for the following outcomes:

- Approve
- Reject
- Yes
- No
- OK

- Defer
- Accept
- Acknowledge
- Dynamic assignment function names:

Labels for dynamic assignment functions. These strings are returned from the runtime config service methods `getUserDynamicAssignmentFunctions` and `getGroupDynamicAssignmentFunctions`. The shipped resource bundle contains labels for the standard dynamic assignment functions (`ROUND_ROBIN`, `LEAST_BUSY`, and `MOST_PRODUCTIVE`). If additional custom dynamic assignment functions have been created, then modify the `WorkflowLabels.properties` resource bundle to provide resource strings for the new functions.
- Standard view names:

Labels for standard views. If you want translated resource strings for any standard views you create, then add them here. Standard view resource strings are looked up from the resource bundle, and are returned as the standard view name from the `UserMetadataService` methods `getStandardTaskViewList` and `getStandardTaskViewDetails`. The key for the resource string should be the name given to the standard view when it is created. If no resource string is added for a particular standard view, then the name as entered is used instead.
- Notification messages:

Resource strings used when the task service sends automatic notifications. These can be customized to suit user requirements.
- Task routing error comments:

When an error is encountered in the routing of a task, the task service automatically appends comments to the task to describe the error. The various strings used for the comments are defined in this resource bundle.

A copy of the `WorkflowLabels.properties` resource bundle is available in the sample `workflow-110-workflowCustomizations`.

You can customize the `WorkflowLabels.properties` resource bundle by editing it and then adding the customized version to the class path for workflow services, ahead of the version that ships with the product.

This can be done in the following ways:

- Place the customized file in a directory tree:

`directory_path/oracle/bpel/services/workflow/resource/WorkflowLabels.properties`
- Update the `worklflowCustomClasspathURL` configuration parameter to point to `directory_path`. (As this is a URL, it is possible to host the resource bundles on a web server, and make them accessible to multiple Oracle WebLogic Servers.) This approach is described in detail in sample `workflow-110-workflowCustomizations`. To download this sample, visit the following URL:

<https://soasamples.samplecode.oracle.com/>

31.5.3 Worklist Client Resource Bundles

The ADF worklist client application uses two resource bundles that contain all the strings displayed in the worklist client web application.

- `oracle.bpel.worklistapp.resource.WorkflowResourceBundle:`
This contains strings used by both the ADF Oracle BPM Worklist, and the JSP-based sample Oracle BPM Worklist that shipped with version 10.1.3 of Oracle SOA Suite.
- `oracle.bpel.worklistapp.resource.WorklistResourceBundle:`
This contains strings used only by the ADF Oracle BPM Worklist.

Copies of the worklist resource bundles are available in the sample `workflow-110-workflowCustomizations`.

The sample illustrates how to customize Oracle BPM Worklist by recompiling these resource bundles, and adding the updated classes to Oracle BPM Worklist.

31.5.4 Task Detail ADF Task Flow Resource Bundles

The ADF task flow applications and associated data controls that get created to display the details of a particular task type use the resource bundle `oracle.bpel.services.workflow.worklist.resource.worklist` to store their resource strings.

You can provide your own custom resource strings for a task detail ADF task flow by adding a customized resource bundle in the task flow application.

You can localize the XML element name displayed in the task flow form through this resource bundle. You can add keys, and use them in the task flow form contents section. The input text label looks as follows:

```
{resources.mykeyword}
```

A copy of the `WorkflowLabels.properties` resource bundle is available in the sample `workflow-110-workflowCustomizations`. This sample illustrates in detail how to provide your own customized resource strings for the task detail ADF task flow application.

31.5.5 Specifying Stage and Participant Names in Resource Bundles

You can provide translated values for stage names and participant names in the composite resource bundle. The resource bundle should contain entries such as the following:

- `stage_name=translated_value`
- `participant_name=translated_value`

31.5.6 Case Sensitivity in Group and Application Role Names

By default, the human workflow system is case insensitive to user names. All user names are stored in lowercase. However, group names and application role names are always case sensitive. User name case insensitivity can be changed in Oracle Enterprise Manager Fusion Middleware Control.

Caution: Only change this setting after performing a new installation. Changing this value on an installation that is actively processing instances, or has many instances in the database, causes serious issues.

To change case sensitivity:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control.
2. In the navigator, expand the **SOA** folder.
3. Right-click **soa-infra**, and select **Administration > System Mbean Browser**.
The System MBean Browser displays on the right side of the page.
4. Expand **Application Defined MBeans > oracle.as.soainfra.config > Server: *server_name* > WorkflowIdentityConfig > human-workflow > WorkflowIdentityConfig.PropertyType**.
5. Click **caseSensitive**.
6. Click the **Operations** tab.
7. Click **setValue**.
8. In the **Value** field, enter **true**, and click **Invoke**.

If you are upgrading from 10.1.3, which by default was case sensitive, set **caseSensitive** to **true** for the system to be the same as with 10.1.3.

31.6 Introduction to Human Workflow Client Integration with Oracle WebLogic Server Services

This section describes how human workflow clients integrate with Oracle WebLogic Server services.

31.6.1 Human Workflow Services Clients

Human workflow services expose the following workflow services:

- Task service
- Task query service
- User metadata service
- Task evidence service
- Task metadata service
- Runtime config service
- Task report service

To use any of these services, you must use the abstract factory pattern for workflow services. The abstract factory pattern provides a way to encapsulate a group of individual factories that have a common theme.

Perform the following tasks:

- Get the `IWorkflowServiceClient` instance for the specific service type. The `WorkflowServiceClientFactory` provides a static factory method to get `IWorkflowServiceClient` according to the service type.

- Use the `IWorkflowServiceClient` instance to get the service instance to use.

The supported service types are Remote and Soap.

Remote clients use Enterprise JavaBeans clients (remote Enterprise JavaBeans, accordingly). SOAP uses SOAP clients. Each type of service requires you to configure workflow clients. [Example 31-7](#) provides details.

Example 31-7 Client Configuration File

```
<workflowServicesClientConfiguration>
<server name="default" default="true">
  <remoteClient>
    <serverURL>t3://myhost.us.oracle.com:7001</serverURL>
    <userName>weblogic</userName>
    <password>weblogic</password>
    <initialContextFactory>weblogic.jndi.WLInitialContextFactory
      </initialContextFactory>
    <participateInClientTransaction>false</participateInClientTransaction>
  </remoteClient>
  <soapClient>
    <rootEndPointURL>http://myhost.us.oracle.com:7001</rootEndPointURL>
    <identityPropagation mode="dynamic" type="saml">
    <policy-references>
      <policy-reference enabled="true" category="security"
        uri="oracle/wss10_saml_token_client_policy"/>
    </policy-references>
    </identityPropagation>
  </soapClient>
</server>
</workflowServicesClientConfiguration>
```

The client configuration file can contain definitions for several configurations. Each server must have its own unique name. If the configuration file defines multiple servers, one server must be set with the default attribute equal to `true`. The `workflowServicesClientConfiguration` has an optional attribute named `serverType` that can be set to one of the following: `LOCAL`, `REMOTE`, or `SOAP`. Each server can override the client type by using the optional attribute `clientType`.

[Example 31-8](#) provides details.

Example 31-8 Client Configuration File with Multiple Configuration Definitions

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<workflowServicesClientConfiguration
  xmlns="http://xmlns.oracle.com/bpel/services/client" clientType="REMOTE"
  <server name="server1" default="true" clientType="SOAP">
    <remoteClient>
      <serverURL>t3://myhost1.us.oracle.com:7001</serverURL>

    <initialContextFactory>weblogic.jndi.WLInitialContextFactory</initialContextFactor
y>

      <participateInClientTransaction>false</participateInClientTransaction>
    </remoteClient> -->
    <soapClient>
      <rootEndPointURL>http://myhost1.us.oracle.com:7001</rootEndPointURL>
      <identityPropagation mode="dynamic" type="saml">
        <policy-references>
          <policy-reference enabled="true" category="security"
            uri="oracle/wss10_saml_token_client_policy"/>
        </policy-references>
```

```

        </identityPropagation>
    </soapClient>
</server>
<server name="server2">
    <remoteClient>
        <serverURL>t3://myhost2.us.oracle.com:7001</serverURL>

<initialContextFactory>weblogic.jndi.WLInitialContextFactory</initialContextFactor
y>
        <participateInClientTransaction>false</participateInClientTransaction>
    </remoteClient> -->
    <soapClient>
        <rootEndPointURL>http://myhost2us.oracle.com:7001</rootEndPointURL>
        <identityPropagation mode="dynamic" type="saml">
            <policy-references>
                <policy-reference enabled="true" category="security"
                    uri="oracle/wss10_saml_token_client_policy"/>
            </policy-references>
        </identityPropagation>
    </soapClient>
</server>
</workflowServicesClientConfiguration>

```

In [Example 31–8](#), server2 uses the default clientType of REMOTE, while server1 overrides the default clientType value to use the clientType of SOAP. The same rule applies if the JAXB WorkflowServicesClientConfigurationType object is used instead of the wf_client_config.xml file.

If the configuration defines a client type, the factory method from WorkflowServiceClientFactory class can be used. [Example 31–9](#) provides details.

Example 31–9 Factory Method from WorkflowServiceClientFactory Class

```

public static IWorkflowServiceClient
getWorkflowServiceClient(WorkflowServicesClientConfigurationType wsc, Logger
logger) throws WorkflowException

```

If the map defines a client type with the property CONNECTION_PROPERTY.CLIENT_TYPE, the factory method in [Example 31–10](#) can be used:

Example 31–10 Factory Method for CONNECTION_PROPERTY.CLIENT_TYPE

```

public static IWorkflowServiceClient getWorkflowServiceClient(Map<CONNECTION_
PROPERTY, String> properties, String serverName, Logger logger) throws
WorkflowException

```

31.6.1.1 Task Query Service Client Code

[Example 31–11](#) provides an example of the task query service client code.

Example 31–11 Task Query Service Client Code

```

/**
 * WFCClientSample
 */
package oracle.bpel.services.workflow.samples;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

```

```

import java.util.Map;

import oracle.bpel.services.workflow.IWorkflowConstants;
import oracle.bpel.services.workflow.WorkflowException;
import oracle.bpel.services.workflow.client.IWorkflowServiceClient;
import oracle.bpel.services.workflow.client.WorkflowServiceClientFactory;
import oracle.bpel.services.workflow.client.IWorkflowServiceClientConstants
    .CONNECTION_PROPERTY;
import oracle.bpel.services.workflow.query.ITaskQueryService;
import oracle.bpel.services.workflow.query.ITaskQueryService.AssignmentFilter;
import oracle.bpel.services.workflow.query.ITaskQueryService.OptionalInfo;
import oracle.bpel.services.workflow.repos.Ordering;
import oracle.bpel.services.workflow.repos.Predicate;
import oracle.bpel.services.workflow.repos.TableConstants;
import oracle.bpel.services.workflow.verification.IWorkflowContext;

public class WFClientSample {

    public static List runClient(String clientType) throws WorkflowException {
        try {

            IWorkflowServiceClient wfSvcClient = null;
            ITaskQueryService taskQuerySvc = null;
            IWorkflowContext wfCtx = null;

            // 1. this step is optional since configuration can be set in wf_client_
            //      config.xml file
            Map<CONNECTION_PROPERTY, String> properties = new HashMap<CONNECTION_
PROPERTY, String>();
            if (WorkflowServiceClientFactory.REMOTE_CLIENT.equals(clientType)) {
                properties.put(CONNECTION_PROPERTY.EJB_INITIAL_CONTEXT_FACTORY,
                    "weblogic.jndi.WLInitialContextFactory");
                properties.put(CONNECTION_PROPERTY.EJB_PROVIDER_URL,
                    "t3://myhost.us.oracle.com:7001");
                properties.put(CONNECTION_PROPERTY.EJB_SECURITY_CREDENTIALS,
                    "weblogic");
                properties.put(CONNECTION_PROPERTY.EJB_SECURITY_PRINCIPAL, "weblogic");
            } else if (WorkflowServiceClientFactory.SOAP_CLIENT.equals(clientType)) {
                properties.put(CONNECTION_PROPERTY.SOAP_END_POINT_ROOT,
                    "http://myhost:7001");
                properties.put(CONNECTION_PROPERTY.SOAP_IDENTITY_
PROPAGATION, "non-saml"); // optional
            }
            // 2. gets IWorkflowServiceClient for specified client type
            wfSvcClient =
WorkflowServiceClientFactory.getWorkflowServiceClient(clientType, properties,
null);

            // 3. gets ITaskQueryService instance
            taskQuerySvc = wfSvcClient.getTaskQueryService();

            // 4. gets IWorkflowContext instance
            wfCtx = taskQuerySvc.authenticate("jcooper", "welcome1".toCharArray(),
                "jazzn.com");

            // 5. creates displayColumns
            List<String> displayColumns = new ArrayList<String>(8);
            displayColumns.add("TASKID");
            displayColumns.add("TASKNUMBER");
            displayColumns.add("TITLE");

```

```
        displayColumns.add("CATEGORY");

        // 6. creates optionalInfo
        List<ITaskQueryService.OptionalInfo> optionalInfo = new
ArrayList<ITaskQueryService.OptionalInfo>();
        optionalInfo.add(ITaskQueryService.OptionalInfo.DISPLAY_INFO);

        // 7. creates assignmentFilter
        AssignmentFilter assignmentFilter = AssignmentFilter.MY_AND_GROUP;

        // 8. creates predicate
        List<String> stateList = new ArrayList<String>();
        stateList.add(IWorkflowConstants.TASK_STATE_ASSIGNED);
        stateList.add(IWorkflowConstants.TASK_STATE_INFO_REQUESTED);
        Predicate predicate = new Predicate(TableConstants.WFTASK_STATE_COLUMN,
Predicate.OP_IN, stateList);

        // 9. creates ordering
        Ordering ordering = new Ordering(TableConstants.WFTASK_DUEDATE_COLUMN,
true, false);
        ordering.addClause(TableConstants.WFTASK_CREATEDDATE_COLUMN, true,
false);

        // 10. calls service - query tasks
        List taskList = taskQuerySvc.queryTasks(wfCtx,
                                                (List<String>) displayColumns,
                                                (List<OptionalInfo>) optionalInfo,
                                                (AssignmentFilter)
                                                    assignmentFilter,
                                                (String) null, // keywords is
optional (see javadoc)
// optional
                                                predicate,
                                                ordering,
                                                0, // starting row
100); // ending row for paging, 0
                                                if no paging

        // Enjoy result
        System.out.println("Successfully get list of tasks for client type: " +
clientType +
                                ". The list size is " + taskList.size());
        return taskList;
    } catch (WorkflowException e) {
        System.out.println("Error occurred");
        e.printStackTrace();
        throw e;
    }
}

public static void main(String args[]) throws Exception {
    runClient(WorkflowServiceClientFactory.REMOTE_CLIENT);
    runClient(WorkflowServiceClientFactory.SOAP_CLIENT);
}
}
```


31.6.1.2 Configuration Option

Each type of client is required to have a workflow client configuration. You can set the configuration in the following locations:

- JAXB object
- wf_client_config.xml file
- Property map

The property map is always complementary to the wf_client_config.xml file. The JAXB object or property map can overwrite the configuration attribute. The file is optional. If it cannot be found in the application class path, then the property map is the main source of configuration.

31.6.1.2.1 JAXB Object You can use the JAXB object to define the client configuration. [Example 31–12](#) shows how to use the WorkflowServiceClientFactory method.

Example 31–12 JAXB Object

```
public static IWorkflowServiceClient getWorkflowServiceClient(String clientType,
    WorkflowServicesClientConfigurationType wsc,
    Logger logger) throws WorkflowException
```

31.6.1.2.2 Workflow Client Configuration File - wf_client_config.xml The client configuration XSD schema is present in the wf_client_config.xsd file.

The server configuration should contain three types of clients:

- localClient
- remoteClient
- soapClient

Oracle recommends that you specify all clients. This is because some services (for example, the identity service) do not have remote clients. Therefore, when you use remote clients for other services, the identity service uses the SOAP service.

An example of a client configuration XML file is shown in [Example 31–13](#). The configuration defines a server named default. The XML file must go into the client application's EAR file.

Example 31–13 Client Configuration

```
<workflowServicesClientConfiguration>
  server name="default" default="true">

  <remoteClient>
    <serverURL>t3://myhost.us.oracle.com:7001</serverURL>
    <userName>weblogic</userName>
    <password>weblogic</password>
    <initialContextFactory>weblogic.jndi.WLInitialContextFactory
      </initialContextFactory>
    <participateInClientTransaction>false</participateInClientTransaction>
  </remoteClient>

  <soapClient>
    <rootEndPointURL>http://myhost.us.oracle.com:7001</rootEndPointURL>
    <identityPropagation mode="dynamic" type="saml">
    <policy-references>
      <policy-reference enabled="true" category="security"
```

```
        uri="oracle/wss10_saml_token_client_policy"/>
    </policy-references>
</identityPropagation>
</soapClient>

</server>
</workflowServicesClientConfiguration>
```

You can define client properties in `wf_client_config.xml` when `WorkflowServicesClientConfigurationType wsc` is null.

The `WorkflowServiceClientFactory` `getWorkflowServiceClient()` methods always look for `wf_client_config.xml` in the class path. If this file is found, the client properties are loaded.

All properties defined in either the property map or the JAXB object override values defined in the `wf_client_config.xml` file.

31.6.1.2.3 Workflow Client Configuration in the Property Map To specify the connection property dynamically, you can use a `java.util.Map` to specify the properties. The properties take precedence over definitions in the configuration file. Therefore, the values of the properties overwrite the values defined in `wf_client_config.xml`. If you do not want to dynamically specify connection details to the server, you can omit the property setting in the map and pass a null value to the factory method. In that case, the configuration `wf_client_config.xml` is searched for in the client application class path.

The configuration file must be in the class path only if you want to get the configuration from the file. It is optional to have the file if all settings from the specific client type are done through the property map. The JAXB object is also not required to have the file, since all settings are taken from the JAXB object. [Example 31–14](#) provides details.

Example 31–14 Property Map

```
IWorkflowServiceClient wfSvcClient =
WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory
.REMOTE_CLIENT,
(Map<IWorkflowServiceClientConstants.CONNECTION_PROPERTY, String> ) null, null);
```

If you do so, the value from `wf_client_config.xml` found in the class path is used by the client to access the services. If the file is not found in the class path and you do not provide the setting according to the service type, a workflow exception is thrown. If the properties map is null and the file is not found, an exception is thrown. If the client omits some properties in the map while the file is not found, the service call fails at runtime (the properties are complementary to the file).

You can define client properties by using the `WorkflowServiceClientFactory` method. [Example 31–15](#) provides details.

Example 31–15 WorkflowServiceClientFactory Method

```
public static IWorkflowServiceClient getWorkflowServiceClient(String clientType,
    Map<CONNECTION_PROPERTY, String> properties,
    Logger logger) throws WorkflowException
```

If the map defines a client type with the property `CONNECTION_PROPERTY` type, the factory method in [Example 31–16](#) can be used:

Example 31–16 Factory Method for CONNECTION_PROPERTY Type

```
public static IWorkflowServiceClient getWorkflowServiceClient(Map<CONNECTION_
PROPERTY, String> properties, Logger logger) throws WorkflowException
```

The `IWorkflowServiceClientConstants.CONNECTION_PROPERTY`, which can be used in the properties map for setting client properties, is shown in [Example 31–17](#):

Example 31–17 CONNECTION_PROPERTY

```
public enum CONNECTION_PROPERTY {
    MODE, // not supported , deprecated
    EJB_INITIAL_CONTEXT_FACTORY,
    EJB_PROVIDER_URL,
    EJB_SECURITY_PRINCIPAL,
    EJB_SECURITY_CREDENTIALS,
    // SOAP configuration
    SOAP_END_POINT_ROOT,
    SOAP_IDENTITY_PROPAGATION, // if value is 'saml' then SAML-token
        identity propagation is used
    SOAP_IDENTITY_PROPAGATION_MODE, // "dynamic"
    MANAGEMENT_POLICY_URI, // default value is "oracle/log_policy"
    SECURITY_POLICY_URI, // default value is "oracle/wss10_
        saml_token_client_policy"
    // REMOTE EJB
    TASK_SERVICE_PARTICIPATE_IN_CLIENT_TRANSACTION // default value is
        false
    //(task service EJB starts a new transaction)
    CLIENT_TYPE,
    DISCOVERY_OF_END_POINT,
    WSS_RECIPIENT_KEY_ALIAS,
    EJB_JNDI_SUFFIX // append to jndi name to used foreign jndi name
};
```

Note: If you use the properties map, you do not need to specify `IWorkflowServiceClientConstants.CONNECTION_PROPERTY.MODE`. This property is deprecated in 11g Release 1.

[Example 31–18](#) provides an example for remote Enterprise JavaBeans clients.

Example 31–18 Example for Remote Enterprise JavaBeans Clients

```
Map<CONNECTION_PROPERTY,String> properties = new HashMap<CONNECTION_
PROPERTY,String>();
properties.put(CONNECTION_PROPERTY.EJB_INITIAL_CONTEXT_
FACTORY,"weblogic.jndi.WLInitialContextFactory");

properties.put(CONNECTION_PROPERTY.EJB_PROVIDER_URL,
    "t3://myhost.us.oracle.com:7001");
properties.put(CONNECTION_PROPERTY.EJB_SECURITY_PRINCIPAL, "weblogic");
properties.put(CONNECTION_PROPERTY.EJB_SECURITY_CREDENTIALS, "weblogic");
IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(
        WorkflowServiceClientFactory.REMOTE_CLIENT,
        properties, null);
```

[Example 31–19](#) provides an example for a SOAP client.

Example 31–19 Example for SOAP Client

```
Map<CONNECTION_PROPERTY,String> properties = new HashMap<CONNECTION_
PROPERTY,String>();
properties.put(CONNECTION_PROPERTY.SOAP_END_POINT_ROOT, "http://myhost:7001");
IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(
        WorkflowServiceClientFactory.SOAP_CLIENT,
        properties, null);
```

31.6.1.3 Client Logging

Clients can optionally pass in a `java.util.logging.Logger` to where the client logs messages. If there is no logger specified, the workflow service client code does not log anything. [Example 31–20](#) shows how to pass a logger to the workflow service clients:

Example 31–20 `java.util.logging.Logger`

```
java.util.logging.Logger logger = ....;

IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory
        .REMOTE_CLIENT, properties, logger);
```

31.6.1.4 Configuration Migration Utility

The client configuration schema has changed between release 10.1.3.x and 11g Release 1. To migrate from release 10.1.3.x to 11g Release 1, use the utility shown in [Example 31–21](#).

Example 31–21 Configuration Migration Utility

```
java -classpath wsclient_extended.jar:bpm-services.jar
    oracle.bpel.services.workflow.client.config.MigrateClientConfiguration
    original_file [new_file];
```

where *original_file* is a `wf_client_config.xml` file from 10.1.3.x and *new_file* is the optional name of the new configuration file. If a new name is not specified, the utility backs up the original configuration file and overwrites the `wf_client_config.xml` file.

31.6.2 Identity Propagation

This section describes how to propagate identities using Enterprise JavaBeans and SAML-tokens for SOAP clients.

There are performance implications for getting the workflow context for every request. This is also true for identity propagation. If you use identity propagation with SAML-token or Enterprise JavaBeans, authenticate the client by passing null for the user and password, get the workflow context instance, and use another service call with workflow context without identity propagation.

31.6.2.1 Enterprise JavaBeans Identity Propagation

The client application can propagate user identity to services by using Enterprise JavaBeans identity propagation. The client code is responsible for securing the user identity.

31.6.2.1.1 Client Configuration If you use identity propagation, the client code must omit the element's <userName> and <password> under the <remoteClient> element in the wf_client_config.xml configuration file. In addition, do not populate the following properties into

Map<IWorkflowServiceClientConstants.CONNECTION_PROPERTY,String> properties as you did in [Section 31.6.1.2.3, "Workflow Client Configuration in the Property Map."](#)

- IWorkflowServiceClientConstants.CONNECTION_PROPERTY.EJB_SECURITY_PRINCIPAL
- IWorkflowServiceClientConstants.CONNECTION_PROPERTY.EJB_SECURITY_CREDENTIALS

31.6.2.1.2 Requirements for Client Applications For Identity Propagation Identity propagation only works if the application is deployed under the Oracle WebLogic Server container and secured with container security or the client is secured with a custom JAAS login module.

End users log in to the client application with the correct user name and password. The users using the client application must be available in the identity store used by the SOA application. As a best practice, configure the client to use the same identity store as the workflow services and Oracle SOA Suite are using. This guarantees that if the user exists on the client side, they also exist on the server side.

For information about configuring the identity store, see *Oracle Fusion Middleware Security Guide*.

For information about interacting with custom identity stores, visit the following URL:

http://www.oracle.com/technology/products/id_mgmt/opss/index.html

31.6.2.2 SAML Token Identity Propagation for SOAP Client

If you use a SOAP client, you can use the SAML-token identity propagation supported by Oracle web services.

This section assumes the application resides in and is secured by the Oracle WebLogic Server container.

31.6.2.2.1 Client configuration To enable identity propagation, the client configuration must specify a special propagation mode.

31.6.2.2.2 Identity Propagation Mode Setting Through Properties If properties are used, then populate the property CONNECTION_PROPERTY.SOAP_IDENTITY_PROPAGATION with the value `saml`.

- Dynamic SAML token propagation mode

The SAML token policy is provided dynamically (the default). The property shown in [Example 31–22](#) is optional. If the identity propagation mode is set, you run by default in dynamic mode.

Example 31–22 Identity Propagation Mode Setting Through Properties

```
properties.put(IWorkflowServiceClientConstants.CONNECTION_PROPERTY.SOAP_IDENTITY_PROPAGATION_MODE , "dynamic");
```

By default, SAML-token constructs dynamic policy based on the following security policy URI: `oracle/wss10_saml_token_client_policy`. Logging is not used. To overwrite the default policy URI, the client can add the code shown in [Example 31–23](#).

Example 31–23 Default Policy URI Overwrite

```
properties.put(CONNECTION_PROPERTY.SECURITY_POLICY_URI, "oracle/wss10_saml_token_client_policy");
properties.put(CONNECTION_PROPERTY.MANAGEMENT_POLICY_URI, "oracle/log_policy");
```

[Example 31–24](#) shows the SAML token dynamic client.

Example 31–24 Token Dynamic Client

```
Map<CONNECTION_PROPERTY,String> properties = new HashMap<CONNECTION_PROPERTY,String>();
properties.put(CONNECTION_PROPERTY.SOAP_IDENTITY_PROPAGATION, "saml");
properties.put(CONNECTION_PROPERTY.SOAP_END_POINT_ROOT, "http://myhost.us.oracle.com:7001");
properties.put(CONNECTION_PROPERTY.SECURITY_POLICY_URI, "oracle/wss10_saml_token_client_policy"); //optional
properties.put(CONNECTION_PROPERTY.MANAGEMENT_POLICY_URI, "oracle/log_policy"); //optional
IWorkflowServiceClient client = WorkflowServiceClientFactory.getWorkflowServiceClient(properties, null);
WorkflowServiceClientFactory.SOAP_CLIENT;
```

The client reference to the policy URI must match the server policy URI. Otherwise, SAML token propagation fails.

31.6.2.2.3 Identity Propagation Mode Setting in Configuration File In the configuration file, you can define the propagation mode by using the `<identityPropagation>` element in the `<soapClient>`, as shown in [Example 31–25](#).

Example 31–25 <identityPropagation> Element

```
<soapClient>
  <rootEndPointURL>http://myhost.us.oracle.com:7001</rootEndPointURL>
  <identityPropagation mode="dynamic" type="saml">
    <policy-references>
      <policy-reference enabled="true" category="security"
        uri="oracle/wss10_saml_token_client_policy"/>
    </policy-references>
  </identityPropagation>
</soapClient>
```

For more information, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

31.6.2.2.4 Identity Propagation Mode Setting Through the JAXB Object You can programmatically set the identity propagation mode with the JAXB object.

31.6.2.3 Public Key Alias

You can use the `oracle.wsm.security.util.SecurityConstants.ClientConstants.WSS_RECIPIENT_KEY_ALIAS` property with the workflow client. This property sets the alias for the recipient's public key that is used to encrypt the type outbound message. Use this property to secure workflow services with the public key alias. This property is only relevant when the SOAP client type uses identity propagation.

The client code must add the `WSS_RECIPIENT_KEY_ALIAS` value to the map if the public key alias is defined. [Example 31–26](#) provides details.

Example 31–26 WSS_RECIPIENT_KEY_ALIAS Property

```

Map<CONNECTION_PROPERTY,String> properties = new HashMap<CONNECTION_
PROPERTY,String>();
properties.put(CONNECTION_PROPERTY.SOAP_IDENTITY_PROPAGATION , "saml");
properties.put(CONNECTION_PROPERTY.SOAP_END_POINT_ROOT,
    "http://myhost.us.oracle.com:7001");
properties.put(CONNECTION_PROPERTY.WSS_RECIPIENT_KEY_ALIAS,keyAlias);
// where keyAlias is a key alias value
properties.put(CONNECTION_PROPERTY.SECURITY_POLICY_URI, "oracle/wss10_saml_token_
client_policy"); //optional
properties.put(CONNECTION_PROPERTY.MANAGEMENT_POLICY_URI , "oracle/log_policy");
    //optional
IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(
        WorkflowServiceClientFactory.SOAP_CLIENT,
        properties, null);

```

If the client uses the JAXB `WorkflowServicesClientConfigurationType` object or the `wf_client_config.xml` file, an optional element called `wssRecipientKeyAlias` is added under the `identityPropagation` element for a SOAP client. [Example 31–27](#) provides details.

Example 31–27 wssRecipientKeyAlias Element

```

<xsd:complexType name="identityPropagationType">
    <xsd:sequence>
        <xsd:element name="policy-references" type="PolicyReferencesType"
            minOccurs="0" maxOccurs="1"/>
        <xsd:element name="wssRecipientKeyAlias" type="xsd:string" minOccurs="0"
            maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="type" type="xsd:string" default="saml"/>
    <xsd:attribute name="mode" type="xsd:string" default="dynamic"/>
</xsd:complexType>

```

For more information about how to create and use the public key alias in the credential store, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

31.6.3 Client JAR Files

A client application without identity propagation must have the `bpm-services.jar` file in its class path. For 11g Release 1, the client class path requires the files shown in [Example 31–28](#).

Example 31–28 Client JAR Files

```

${bea.home}/wlserver_10.3/server/lib/wlfullclient.jar
${bea.home}/AS11gR1SOA/webservices/wsclient_extended.jar
${bea.home}/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar
${bea.home}/AS11gR1SOA/soa/modules/oracle.soa.workflow_11.1.1/bpm-services.jar

```

The `wlfullclient.jar` file must be generated.

1. Generate the `wlfullclient.jar` as follows:

```

cd ${bea.home}/wlserver_10.3/server/lib
java -jar ../../modules/com.bea.core.jarbuilder_1.3.0.0.jar

```

31.7 Task States in a Human Task

The following list identifies all the task states available in a human task. The constants for all states are defined in `IWorkflowConstants.java`.

- `String TASK_STATE_ALERTED = "ALERTED";`
- `String TASK_STATE_ASSIGNED = "ASSIGNED";`
- `String TASK_STATE_COMPLETED = "COMPLETED";`
- `String TASK_STATE_DELETED = "DELETED";`
- `String TASK_STATE_ERRORED = "ERRORED";`
- `String TASK_STATE_EXPIRED = "EXPIRED";`
- `String TASK_STATE_INFO_REQUESTED = "INFO_REQUESTED";`
- `String TASK_STATE_OUTCOME_UPDATED = "OUTCOME_UPDATED";`
- `String TASK_STATE_STALE = "STALE";`
- `String TASK_STATE_SUSPENDED = "SUSPENDED";`
- `String TASK_STATE_WITHDRAWN = "WITHDRAWN";`

For more information about `IWorkflowConstants.java`, see *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*.

31.8 Database Views for Oracle Workflow

This section describes database views that enable queries against the Oracle workflow services schema to receive reports. [Table 31–22](#) lists the reports exposed in Oracle BPM Worklist and the database views corresponding to these reports.

Table 31–22 Report Views

Existing Worklist Report	Corresponding Database View
Unattended Tasks report	WFUNATTENDEDTASKS_VIEW
Task Cycle Time report	WFTASKCYCLETIME_VIEW
Task Productivity report	WFPRODUCTIVITY_VIEW
Task Priority Report	WFTASKPRIORITY_VIEW

31.8.1 Unattended Tasks Report View

[Table 31–23](#) describes the `WFUNATTENDEDTASKS_VIEW` report view.

Table 31–23 Unattended Tasks Report View

Name	Type
TASKID ¹	VARCHAR2 (64)
TASKNAME	VARCHAR2 (200)
TASKNUMBER	NUMBER
CREATEDDATE	DATE
EXPIRATIONDATE	DATE
STATE	VARCHAR2 (100)

Table 31–23 (Cont.) Unattended Tasks Report View

Name	Type
PRIORITY	NUMBER
ASSIGNEEGROUPS	VARCHAR2 (2000)

¹ NOT NULL column

For example:

- Query unattended tasks that have an expiration date of next week, as shown in [Example 31–29](#).

Example 31–29 Query of Unattended Tasks with an Expiration Date of Next Week

```
SELECT tasknumber, taskname, assigneegroups FROM WFUNATTENDEDTASKS_VIEW
WHERE expirationdate > current_date AND expirationdate < current_date +
7;
```

- Query unattended tasks for mygroup, as shown in [Example 31–30](#).

Example 31–30 Query of Unattended Tasks for mygroup

```
SELECT tasknumber, taskname, assigneegroups FROM WFUNATTENDEDTASKS_VIEW
WHERE 'mygroup' IN assigneegroups;
```

- Query unattended tasks created in the last 30 days, as shown in [Example 31–31](#).

Example 31–31 Query of Unattended Tasks Created in the Last 30 Days

```
SELECT tasknumber, taskname, assigneegroups FROM WFUNATTENDEDTASKS_VIEW
WHERE createddate > current_date -30;
```

31.8.2 Task Cycle Time Report View

[Table 31–24](#) describes the WFTASKCYCLETIME_VIEW report view.

Table 31–24 Task Cycle Time Report View

Name	Type
TASKID ¹	VARCHAR2 (64)
TASKNAME	VARCHAR2 (200)
TASKNUMBER	NUMBER
CREATEDDATE	DATE
ENDDATE	DATE
CYCLETIME	NUMBER (38)

¹ NOT NULL column

For example:

- Compute the average cycle time (task completion time) for completed tasks that were created in the last 30 days, as shown in [Example 31–32](#).

Example 31–32 Average Cycle Time for Completed Tasks Created in the Last 30 Days

```
SELECT avg(cycletime) FROM WFTASKCYCLETIME_VIEW WHERE createddate >
```

```
(current_date - 30);
```

- Query the average cycle time for all completed tasks created in the last 30 days and group them by task name, as shown in [Example 31–33](#).

Example 31–33 Average Cycle Time for All Completed Tasks Created in Last 30 days Grouped by Task Name

```
SELECT taskname, avg(cycletime) FROM WFTASKCYCLETIME_VIEW WHERE
createddate > (current_date - 30) GROUP BY taskname;
```

- Query the least and most time taken by each task, as shown in [Example 31–34](#).

Example 31–34 least and most time taken by each task

```
SELECT taskname, min(cycletime), max(cycletime) FROM WFTASKCYCLETIME_VIEW
GROUP BY taskname;
```

- Compute the average cycle time for tasks completed in the last seven days, as shown in [Example 31–35](#).

Example 31–35 Average Cycle Time for Tasks Completed in the Last Seven Days

```
SELECT avg(cycletime) FROM WFTASKCYCLETIME_VIEW WHERE enddate >
(current_date - 7);
```

- Query tasks that took more than seven days to complete, as shown in [Example 31–36](#).

Example 31–36 Tasks Taking More than Seven Days to Complete

```
SELECT taskname, avg(cycletime) FROM WFTASKCYCLETIME_VIEW WHERE cycletime
> ((current_date +7) - current_date) GROUP BY taskname;
```

31.8.3 Task Productivity Report View

[Table 31–25](#) describes the WFPRODUCTIVITY_VIEW report view.

Table 31–25 Task Productivity Report View

Name	Type
TASKNAME	VARCHAR2 (200)
TASKID	VARCHAR2 (200)
TASKNUMBER	NUMBER
USERNAME	VARCHAR2 (200)
STATE ¹	VARCHAR2 (100)
LASTUPDATEDDATE	DATE

¹ For completed tasks, the state is null. Use decode(outcome, '', 'COMPLETED', outcome) in queries.

For example:

- Count the number of unique tasks that the user has updated in the last 30 days, as shown in [Example 31–37](#).

Example 31–37 Number of Unique Tasks Updated in the Last 30 Days

```
SELECT username, count(distinct(taskid)) FROM WFPRODUCTIVITY_VIEW WHERE
lastupdateddate > (current_date -30) GROUP BY username;
```

- Count the number of tasks that the user has updated (one task may have been updated multiple times) in the last seven days, as shown in [Example 31–38](#).

Example 31–38 Number of Tasks Updated in the Last 7 Days

```
SELECT username, count(taskid) FROM WFPRODUCTIVITY_VIEW WHERE
lastupdateddate > (current_date -7) GROUP BY username;
```

- Count the number of tasks of each task type on which the user has worked, as shown in [Example 31–39](#).

Example 31–39 Number of Tasks of Each Task Type on Which the User has Worked

```
SELECT username, taskname, count(taskid) FROM WFPRODUCTIVITY_VIEW GROUP
BY username, taskname;
```

- Count the number of tasks of each task type that the user has worked on in the last 100 days, as shown in [Example 31–40](#).

Example 31–40 Number of Tasks of Each Task Type Worked on in the Last 100 Days

```
SELECT username, taskname, count(taskid) FROM WFPRODUCTIVITY_VIEW WHERE
lastupdateddate > (current_date -100) GROUP BY username, taskname;
```

31.8.4 Task Priority Report View

[Table 31–26](#) describes the WFTASKPRIORITY_VIEW report view.

Table 31–26 Task Priority Report View

Name	Type
TASKID ¹	VARCHAR2 (64)
TASKNAME	VARCHAR2 (200)
TASKNUMBER	NUMBER
PRIORITY	NUMBER
OUTCOME	VARCHAR2 (100)
ASSIGNEDDATE	DATE
UPDATEDDATE	DATE
UPDATEDBY	VARCHAR2 (64)

¹ NOT NULL column

For example:

- Query the number of tasks updated by each user in each task priority, as shown in [Example 31–41](#).

Example 31–41 Number of Tasks Updated by Each User in Each Task Priority

```
SELECT updatedby, priority, count(taskid) FROM WFTASKPRIORITY_VIEW GROUP
BY updatedby, priority;
```

- Query task-to-outcome distribution, as shown in [Example 31–42](#).

Example 31–42 Task-to-outcome Distribution

```
SELECT taskname, decode(outcome, '', 'COMPLETED', outcome), count
(taskid) FROM WFTASKPRIORITY_VIEW GROUP BY taskname, outcome;
```

- Query the number of tasks updated by the given user in each priority, as shown in [Example 31–43](#).

Example 31–43 Number of Tasks Updated by the Given User in Each Priority

```
SELECT priority, count(taskid) FROM WFTASKPRIORITY_VIEW WHERE
updatedby='jstein' GROUP BY priority;
```

Integrating Microsoft Excel with a Human Task

This chapter describes how to integrate the enterprise system capabilities of Oracle SOA Suite with Microsoft Excel 2007. This integration enables you to invoke a BPEL process from Microsoft Excel and attach Microsoft Excel workbooks to workflow email notifications. You can configure this integration without having to switch between tools.

This chapter includes the following sections:

- [Section 32.1, "Configuring Your Environment for Invoking a BPEL Process from an Excel Workbook"](#)
- [Section 32.2, "Attaching Excel Workbooks to Human Task Workflow Email Notifications"](#)

32.1 Configuring Your Environment for Invoking a BPEL Process from an Excel Workbook

From an Excel workbook, you can invoke a BPEL process that is deployed in Oracle WebLogic. To perform this task, you install a plug-in of the Application Development Framework Desktop Integration (ADF-DI) on the same host as the Excel document that invokes the BPEL process.

To enable this functionality, do the following:

32.1.1 How to Create an JDeveloper Project of the Type Web Service Data Control

You use the Create Web Service Data Control Wizard to create the project.

To create an Oracle JDeveloper project of the type web service data control:

1. In JDeveloper, from the **File** menu, select **New**. The New Gallery dialog appears.
2. In the **Categories** section, expand **Business Tier**, then select **Data Controls**. The corresponding items appear in the **Items** pane.
3. In the **Items** pane, select **Web Service Data Control** and click **OK**. The Create Web Service Data Control Wizard appears.
4. Follow the instructions in the online Help for this wizard. As you follow these instructions, you are prompted to select the WSDL file and operations to use for this project.

32.1.2 How to Create a Dummy JSF Page

In this task you generate a page definition file. Note that the actual layout generated in the JSF file is not of a concern. Instead, you simply want to generate a page definition file that contains these controls and actions. This page definition is used later in the Excel file.

To create a dummy JSF page:

1. In JDeveloper, from the **File** menu, select **New**. The New Gallery dialog appears.
2. In the **Categories** section, from the **Web Tier** node, select **JSF**. The corresponding items appear in the Items pane.
3. In the **Items** pane, select **JSF Page** and then click **OK**. The Create JSF Page dialog appears.
4. Fill in the various fields by following the instructions in the online Help for this dialog.
5. When prompted, drag and drop from the Component Palette the controls and fields you are interested in using in the Excel document.

For an example of how to perform this task, see ["Task 3: Create a Valid Page Definition File to Be Used in the Excel Workbook"](#) on page 32-13.

32.1.3 How to Add Desktop Integration to Your Oracle JDeveloper Project

To add Oracle ADF Desktop Integration to the technology scope of your project, use the Project Properties dialog in JDeveloper.

To add Oracle ADF Desktop Integration to your project:

1. In the Application Navigator, right-click the project to which you want to add the Oracle ADF Desktop Integration module and choose **Project Properties** from the context menu.

If your application uses the Fusion Web Application (ADF) application template, you select the **ViewController** project. If your application uses another application template, select the project that corresponds to the web application.

2. In the Project Properties dialog, select **Technology Scope** to view the list of available technologies.
3. Choose the **ADF Desktop Integration** and **ADF Library Web Application Support** project technologies and add them to the **Selected Technologies** list.
4. Click **OK**.

32.1.4 What Happens When You Add Desktop Integration to Your JDeveloper Project

When you add the Oracle ADF Desktop Integration module to the technology scope of your project, the following events occur:

- The project adds the Oracle ADF Desktop Integration runtime library. This library references the following `.jar` files in its class path:
 - `wsclient.jar`
 - `adf-desktop-integration.jar`
 - `resourcebundle.jar`
- The project adds an ADF bindings filter (`adfBindings`).

- The project's deployment descriptor (`web.xml`) is modified to include the following entries:
 - A servlet named `adfdiRemote`

Note: The value for the `url-pattern` attribute of the `servlet-mapping` element for `adfdiRemote` must match the value of the `RemoteServletPath` workbook property described in *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

- A filter named `adfdiExcelDownload`
- A MIME mapping for Excel files (`.xlsx` and `.xlsm`)

The previous list is not exhaustive. Adding Oracle ADF Desktop Integration to a project makes other changes to `web.xml`. Note that some entries in `web.xml` are only added if they do not exist.

When you add ADF Library Web Application Support to the technology scope of your project, the project's `web.xml` file is modified to include the entries shown in [Example 32-1](#).

Example 32-1 *web.xml File Entries*

```
<filter>
  <filter-name>ADFLibraryFilter</filter-name>
  <filter-class>oracle.adf.library.webapp.LibraryFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>ADFLibraryFilter</filter-name>
  <url-pattern>*/</url-pattern>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<servlet>
  <servlet-name>adflibResources</servlet-name>
  <servlet-class>oracle.adf.library.webapp.ResourceServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>adflibResources</servlet-name>
  <url-pattern>/adflib/*</url-pattern>
</servlet-mapping>
```

Make sure that the filter for ADF Library Web Application Support (`<filter-name>ADFLibraryFilter</filter-name>`) appears below the `adfdiExcelDownload` filter entries in `web.xml` as shown in [Example 32-2](#) so that integrated Excel workbooks can be downloaded from the Fusion web application.

Example 32-2 *web.xml File Entries*

```
<filter>
  <filter-name>adfdiExcelDownload</filter-name>
  <filter-class>oracle.adf.desktopintegration.filter.DIExcelDownloadFilter</filter-class>
</filter>
<filter>
  <filter-name>ADFLibraryFilter</filter-name>
  <filter-class>oracle.adf.library.webapp.LibraryFilter</filter-class>
```

```
</filter>
...
<filter-mapping>
<filter-name>adfdiExcelDownload</filter-name>
<url-pattern>*.xlsx</url-pattern>
</filter-mapping>
<filter-mapping>
<filter-name>adfdiExcelDownload</filter-name>
<url-pattern>*.xlsm</url-pattern>
</filter-mapping>
...
<filter-mapping>
<filter-name>ADFLibraryFilter</filter-name>
<url-pattern>*/*</url-pattern>
<dispatcher>FORWARD</dispatcher>
<dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

For more information about `web.xml`, see *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

32.1.5 How to Deploy the Web Application You Created in Step 1

For an example of how to perform this task, see [Section 32.2.3.5, "Task 5: Deploy the ADF Task Flow."](#)

32.1.6 How to Install Microsoft Excel

Install Microsoft Excel by following the appropriate Microsoft documentation.

32.1.7 How to Install the Oracle ADF-Desktop Integration Plug-in

To perform this installation, follow the steps in [Section 32.2.3.4, "Task 4: Prepare the Excel Workbook."](#)

32.1.8 How to Specify the User Interface Controls and Create the Excel Workbook

For instructions see [Section 32.2.3.4, "Task 4: Prepare the Excel Workbook."](#)

32.2 Attaching Excel Workbooks to Human Task Workflow Email Notifications

As an alternative to using Oracle BPM Worklist, you can attach an Excel workbook with task details as part of a Human Task workflow email notification. In this case, the user receives an email about a new task. This email has an Excel workbook attached, and, when the user opens the attachment, they are directed to a login page--similar to that for Oracle BPM Worklist. The Excel workbook includes such task details as task ID, payload, and so on. Buttons correspond to the actions the user can perform, and clicking one of them invokes the corresponding BPEL process.

32.2.1 Enabling Attachment of Excel Workbooks to Human Task Workflow Email Notifications

To enable this functionality, do the following:

1. In Oracle JDeveloper, create an ADF task flow that corresponds to a particular human task activity in a BPEL process.
2. Modify the settings in the ADF-DI-enabled Excel sheet to point to the server on which the task flow is deployed, then save this Excel sheet as part of the .war file packaged for the ADF task flow. The steps for doing these things are covered in [Section 32.2.3, "Example: Attaching an Excel Workbook to Email Notifications."](#) Later, you use the page definition files generated in [Section 32.1.2, "How to Create a Dummy JSF Page."](#)

Note: Packaging the Excel workbook with the ADF task flow assumes that there is a one-to-one correspondence between the ADF task flow and the Excel sheet used for a workflow.

3. Enable the ADF task flow project for desktop integration and deploy it to the server.

32.2.2 What Happens During Runtime When You Enable Attachment of Excel Workbooks to Human Task Workflow Email Notifications

Note the following end-user experience during runtime:

- A user receives an email notification regarding a new task, with the Excel attachment. When the attachment is opened, the user is directed to a login page and prompted to enter a username and password. This login page is similar to the login page for Oracle BPM Worklist.
- The Excel workbook loads up with the task details—for example, task identifier, payload, and so on. There are buttons corresponding to actions the user can take. Clicking one of these buttons starts the BPEL process in which the task is a step.

Note the following runtime behaviors:

- The Excel workbook is added as an attachment only when the flag **include task attachments** for the corresponding task is set to `true`.
- Before adding the Excel workbook as an attachment, runtime verifies that a digital signature is not enabled for the workflow.
- When the ADF task flow is deployed to the server, such data as the hostname and port number of the task flow URI is registered in the database.
- When an email notification is created, runtime retrieves from the database the hostname and port number of the application server and the context root of the task flow application. It uses this information to find the Excel workbook, *workflow_name.xls*.

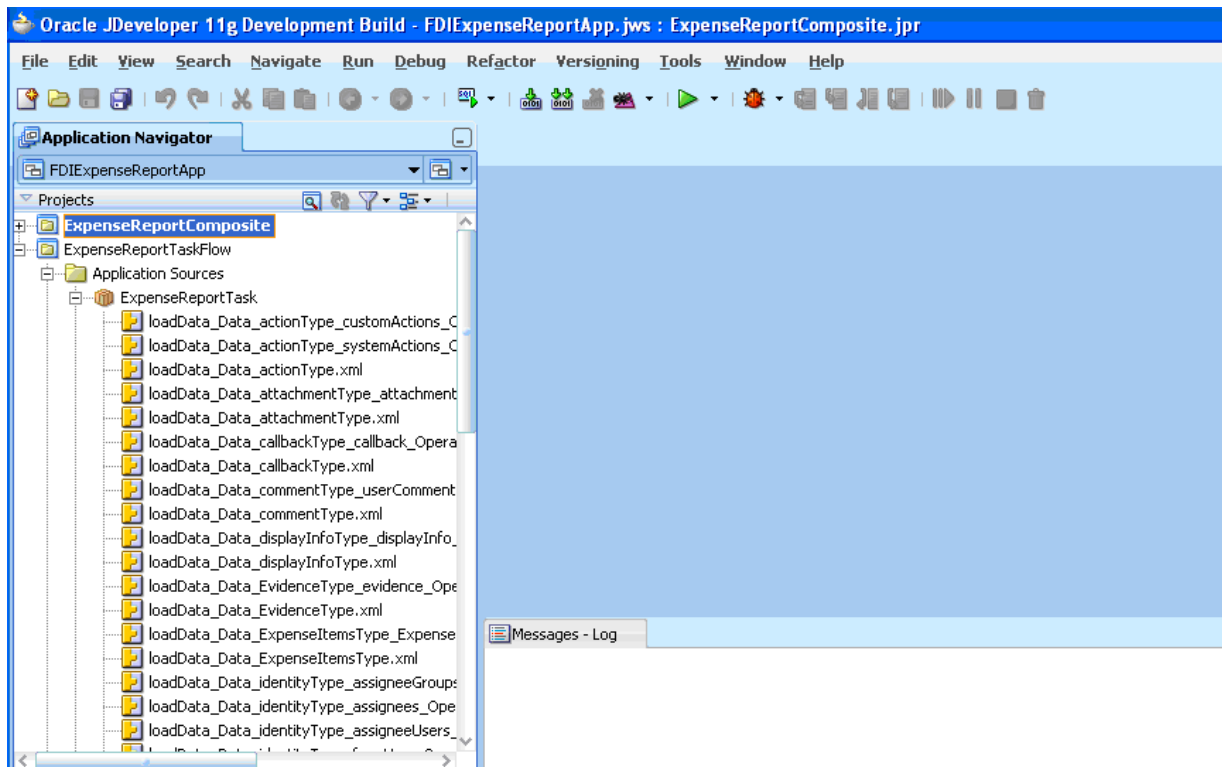
32.2.3 Example: Attaching an Excel Workbook to Email Notifications

This section describes how to attach an Excel workbook to email notifications.

32.2.3.1 Task 1: Enable the ADF Task Flow Project with Oracle ADF-DI Capabilities

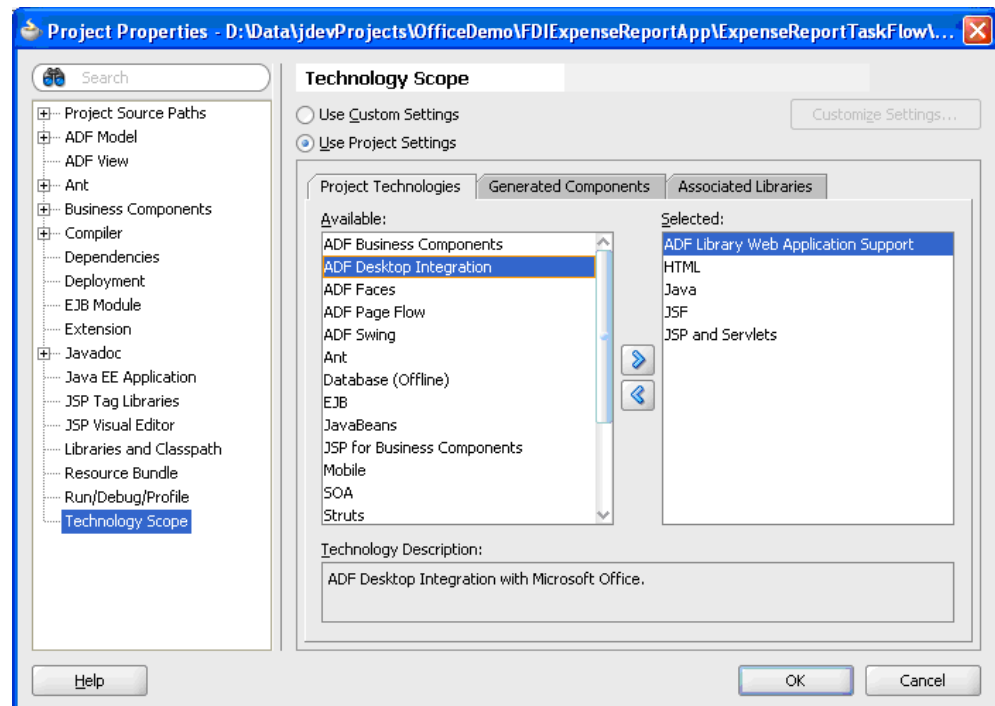
In this task, you configure the web application to work with Oracle ADF-DI.

1. Create an ADF task flow project based on a human task. This creates a data control corresponding to the task, and .xml files corresponding to the task's structure. [Figure 32–1](#) shows Oracle JDeveloper with a sample project open.

Figure 32–1 Oracle JDeveloper with a Sample Project Open

2. Add Oracle ADF Desktop Integration to the project by following the instructions in [Section 32.1.3, "How to Add Desktop Integration to Your Oracle JDeveloper Project."](#)

Figure 32–2 illustrates the Oracle JDeveloper Project Properties dialog when you are adding Oracle ADF Desktop Integration to your project.

Figure 32–2 Oracle JDeveloper Project Properties Dialog

3. When the technology scopes mentioned in Step 2 are added to the project, verify that the necessary events have occurred:
 - a. In the Application Navigator, right-click the project.
 - b. Click **Project Properties**, then select **Libraries and Classpath**.
 - c. Confirm that the entry **ADF Desktop Integration Runtime** exists and is checked.
 - d. Select this library and click **View**.
 - e. Confirm that the library references `wsclient.jar` and `adf-desktop-integration.jar` in its class path.
4. Confirm that the project's deployment descriptor—namely, `web.xml`—is modified to include the following entries:
 - A servlet named `adfdiRemote`
 - A filter named `adfdiExcelDownload`
 - A MIME mapping for Excel files

The previous list is not exhaustive. Adding **ADF Desktop Integration** and **ADF Library Web Application Support** to the project makes other changes to `web.xml`. Here is a sample snippet of the deployment descriptor:

```
<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>client</param-value>
</context-param>
<context-param>
  <description>...</description>
  <param-name>org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION
  </param-name>
  <param-value>>false</param-value>
```

```

</context-param>
<context-param>
    <description>Whether the 'Generated by...' comment at the bottom of ADF
    Faces HTML pages should contain version number information.</description>
    <param-name>oracle.adf.view.rich.versionString.HIDDEN</param-name>
    <param-value>>false</param-value>
</context-param>
<filter>
    <filter-name>trinidad</filter-name>
    <filter-class>org.apache.myfaces.trinidad.webapp.TrinidadFilter
</filter-class>
</filter>
<filter>
    <filter-name>ADFLibraryFilter</filter-name>
    <filter-class>oracle.adf.library.webapp.LibraryFilter
</filter-class>
</filter>
<filter>
    <filter-name>adfBindings</filter-name>
    <filter-class>oracle.adf.model.servlet.ADFBindingFilter
</filter-class>
</filter>
<filter>
    <filter-name>adfdiExcelDownload</filter-name>
    <filter-class>
        oracle.adf.desktopintegration.filter.DIExcelDownloadFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>trinidad</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<filter-mapping>
    <filter-name>adfBindings</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<filter-mapping>
    <filter-name>trinidad</filter-name>
    <servlet-name>adfdiRemote</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>adfBindings</filter-name>
    <servlet-name>adfdiRemote</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>adfdiExcelDownload</filter-name>
    <url-pattern>*.xlsx</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>adfdiExcelDownload</filter-name>
    <url-pattern>*.xlsm</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>ADFLibraryFilter</filter-name>
    <url-pattern>/*</url-pattern>

```

```

        <dispatcher>FORWARD</dispatcher>
        <dispatcher>REQUEST</dispatcher>
    </filter-mapping>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet>
        <servlet-name>resources</servlet-name>

<servlet-class>org.apache.myfaces.trinidad.webapp.ResourceServlet</servlet-clas
s>
</servlet>
<servlet>
        <servlet-name>adflibResources</servlet-name>

        <servlet-class>oracle.adf.library.webapp.ResourceServlet</servlet-class>
    </servlet>
<servlet>
        <servlet-name>adfdiRemote</servlet-name>

<servlet-class>oracle.adf.desktopintegration.servlet.DIRemoteServlet</servlet-c
lass>
</servlet>
<servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
<servlet-mapping>
        <servlet-name>resources</servlet-name>
        <url-pattern>/adf/*</url-pattern>
    </servlet-mapping>
<servlet-mapping>
        <servlet-name>resources</servlet-name>
        <url-pattern>/afr/*</url-pattern>
    </servlet-mapping>
<servlet-mapping>
        <servlet-name>adflibResources</servlet-name>
        <url-pattern>/adflib/*</url-pattern>
    </servlet-mapping>
<servlet-mapping>
        <servlet-name>adfdiRemote</servlet-name>
        <url-pattern>/adfdiRemoteServlet</url-pattern>
    </servlet-mapping>
<session-config>
        <session-timeout>35</session-timeout>
</session-config>
<mime-mapping>
        <extension>html</extension>
        <mime-type>text/html</mime-type>
</mime-mapping>
<mime-mapping>
        <extension>txt</extension>
        <mime-type>text/plain</mime-type>
</mime-mapping>
<mime-mapping>
        <extension>xlsx</extension>

<mime-type>application/vnd.openxmlformats-officedocument.spreadsheetml.sheet</m

```

```
ime-type>
</mime-mapping>
<mime-mapping>
    <extension>xlsm</extension>
    <mime-type>application/vnd.ms-excel.sheet.macroEnabled.12</mime-type>
</mime-mapping>
```

5. Add the following `<auth-filter>` entry to `weblogic.xml`.

```
<weblogic-web-app>
    <auth-filter>oracle.bpel.services.workflow.client.worklist.util.FDIFilter
</auth-filter>
.
.
</weblogic-web-app>
```

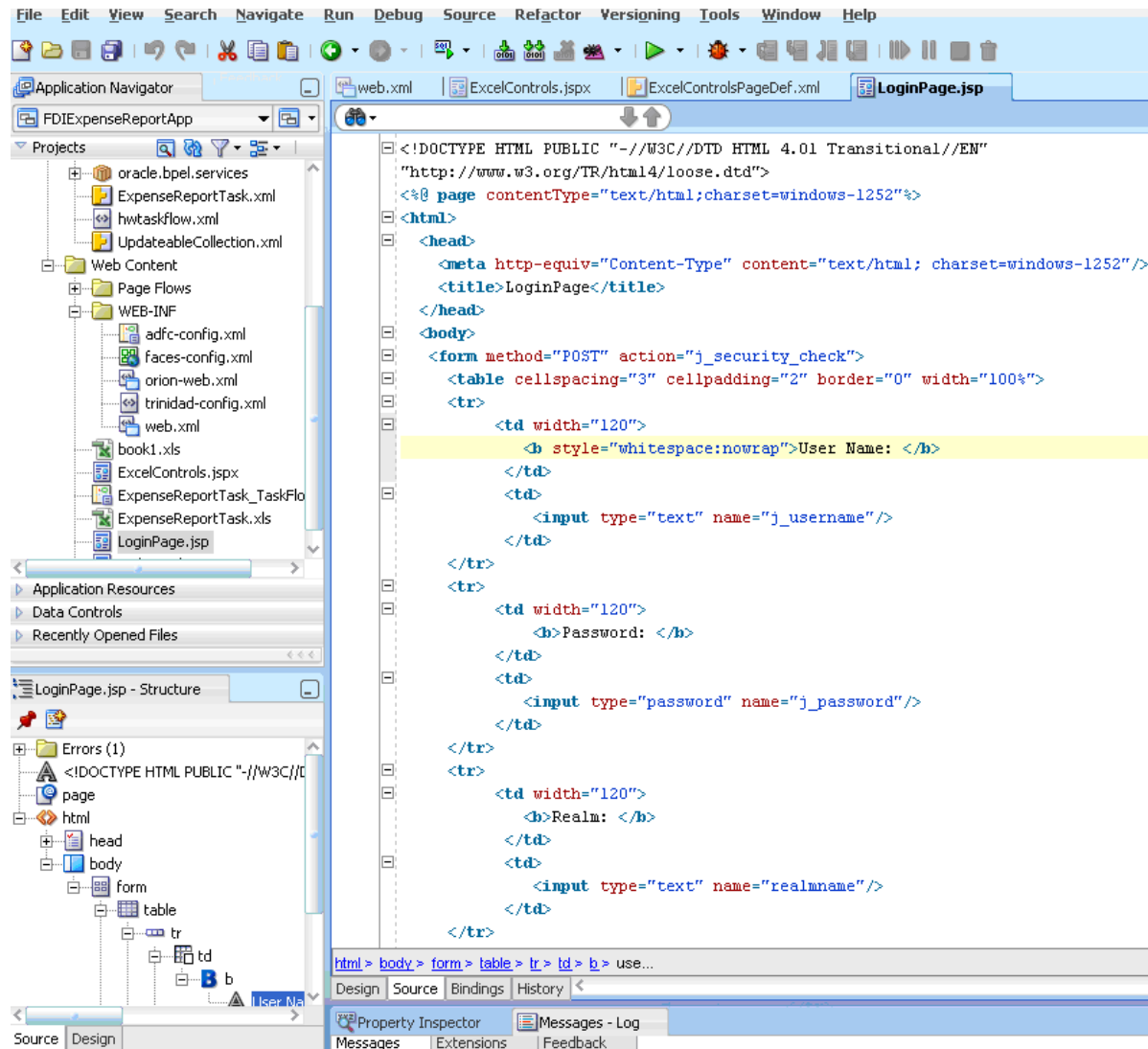
6. Click **Save All**. Right-click the project and click **Rebuild**. Make sure there are no compilation errors and the build completes successfully.

The web application is now configured to work with Oracle ADF-DI.

32.2.3.2 Task 2: Set up Authentication

This task is required to add Oracle ADF-Desktop Integration to create a web session for an Excel workbook.

1. Add ADF security to your project:
 - a. From the **Application** menus, then **Secure**, then **Configure ADF Security**.
 - b. Select **ADF Authentication**.
 - c. Click **Finish**.
2. Create a login page for the application:
 - a. From the directory `ExpenseReportTaskFlow\public_html\`, copy the file `LoginPage.jsp` to the directory `project_home\public_html`.
 - b. Refresh the view in Oracle JDeveloper.
 - c. Verify that the file `LoginPage.jsp` is visible. It should appear as illustrated in [Figure 32-3](#).

Figure 32–3 Oracle JDeveloper: Login.jsp File

3. Once you have added ADF security, confirm that the following entries are added to the `web.xml` file. If some entries are missing, add them manually. Note that form authentication, using the login page created in Step 2 of [Section 32.2.3.2, "Task 2: Set up Authentication,"](#) is used.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>allPages</web-resource-name>
    <url-pattern>/</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>Administrators</role-name>
  </auth-constraint>
</security-constraint>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>adfAuthentication</web-resource-name>
    <url-pattern>/adfAuthentication</url-pattern>
  </web-resource-collection>
  <auth-constraint>
```

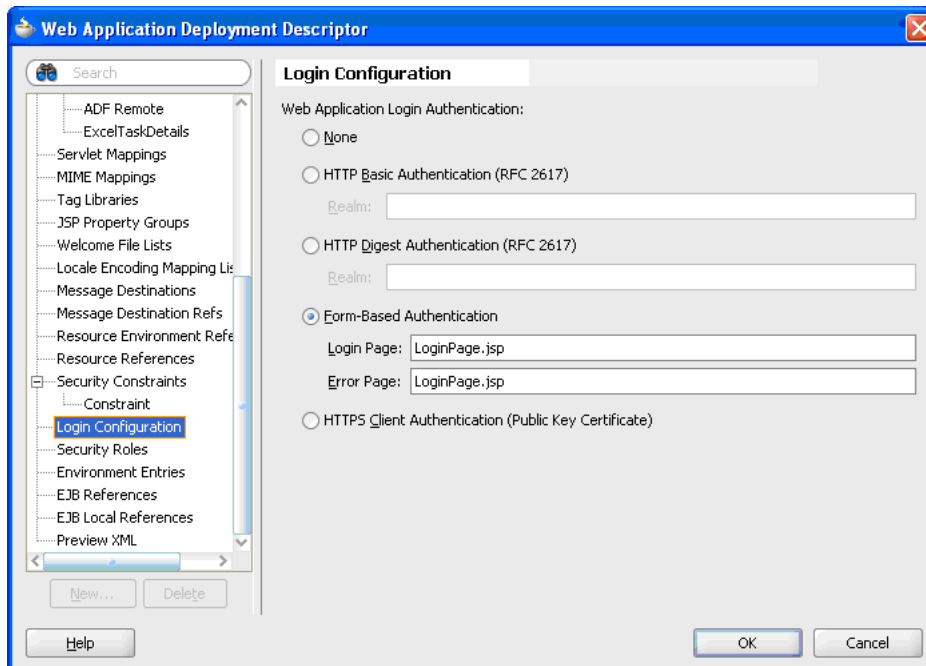
```

        <role-name>Administrators</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>jazn.com</realm-name>
    <form-login-config>
        <form-login-page>/LoginPage.jsp</form-login-page>
        <form-error-page>/LoginPage.jsp</form-error-page>
    </form-login-config>
</login-config>
<security-role>
    <role-name>Administrators</role-name>
</security-role>

```

Figure 32–4 shows how these entries appear graphically in the Web Application Deployment Descriptor dialog.

Figure 32–4 Oracle JDeveloper: Application Deployment Descriptor



4. For every logical security role added in `web.xml`, make a corresponding entry in `weblogic.xml` as follows:

```

<weblogic-web-app>

<auth-filter>oracle.bpel.services.workflow.client.worklist.util.FDIAuthFilter</auth-filter>

  <security-role-assignment>
    <role-name>Administrators</role-name>
    <principal-name>fmwadmin</principal-name>
    <principal-name>users</principal-name>
  </security-role-assignment>
  .
  .
</weblogic-web-app>

```


5. Click **Save All**.

The ADF task flow web application is now configured for login capability that can be used by the Excel workbook.

32.2.3.3 Task 3: Create a Valid Page Definition File to Be Used in the Excel Workbook

The page definition file `ExcelControlsPageDef.xml` is used to create the Excel workbook. Perform the following steps:

1. Create a new Java class by following these steps:
 - a. Select **Technologies**, then select **General**, then select **Simple Files**, then select **Java Class**.

- b. Specify details as follows:

Name: TaskRetrievers

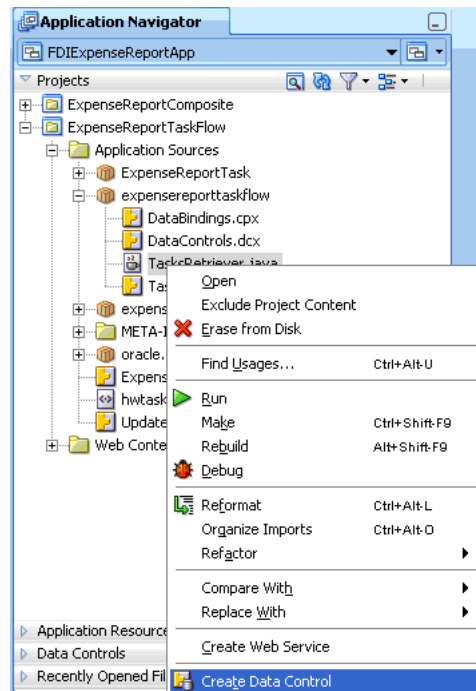
Package: (leave it as default)

Extends: oracle.bpel.services.workflow.client.worklist.excel.TasksRetriever
(Click **Browse** to select this class.)

This creates a new Java class `<default-package>.TasksRetriever`.

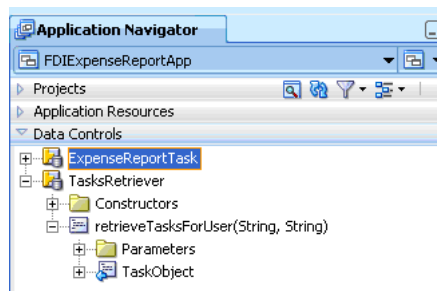
2. Create a data control for this newly created Java class. This data control provides access to an API that retrieves all assigned tasks for a user. [Figure 32–5](#) shows the menu for creating the data control.

Figure 32–5 Oracle JDeveloper: Creating a Data Control



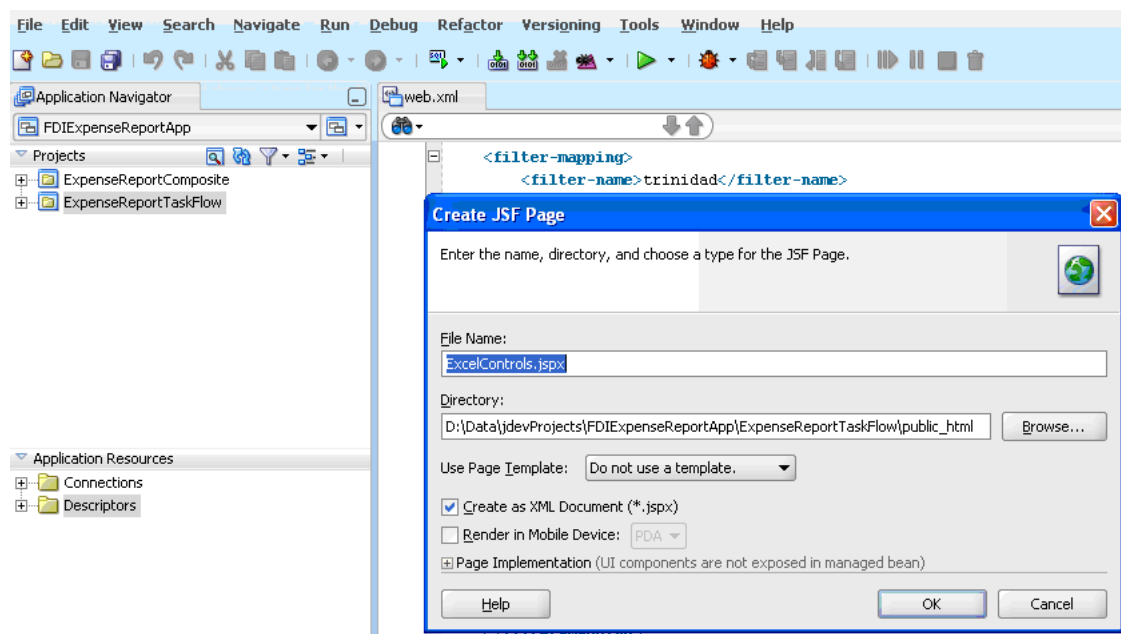
3. Verify that the newly created data control **TasksRetriever** is visible in the Data Control palette in the lower portion of the Application Navigator. [Figure 32–6](#) shows the Application Navigator with the Data Control palette expanded.

Figure 32–6 Oracle JDeveloper: Application Navigator with Data Control Palette Expanded

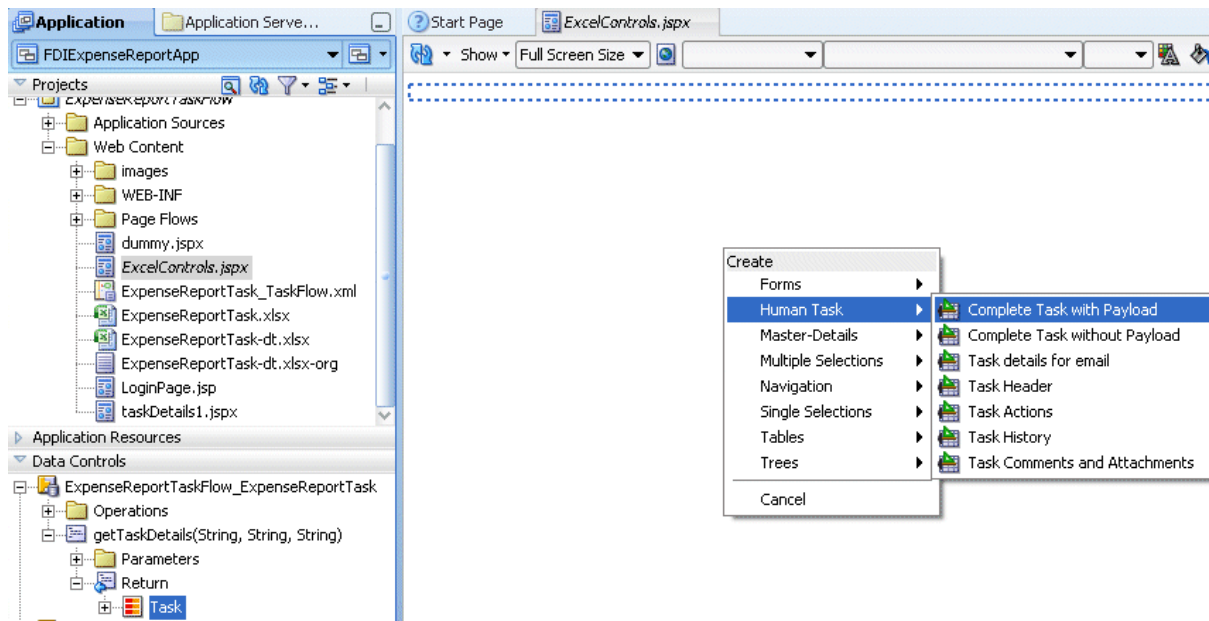


4. Create a new JSF JSP page--namely, `ExcelControls.jspx`. This generates a page definition that can be used by ADF-DI while authoring the Excel document. [Figure 32–7](#) provides details.

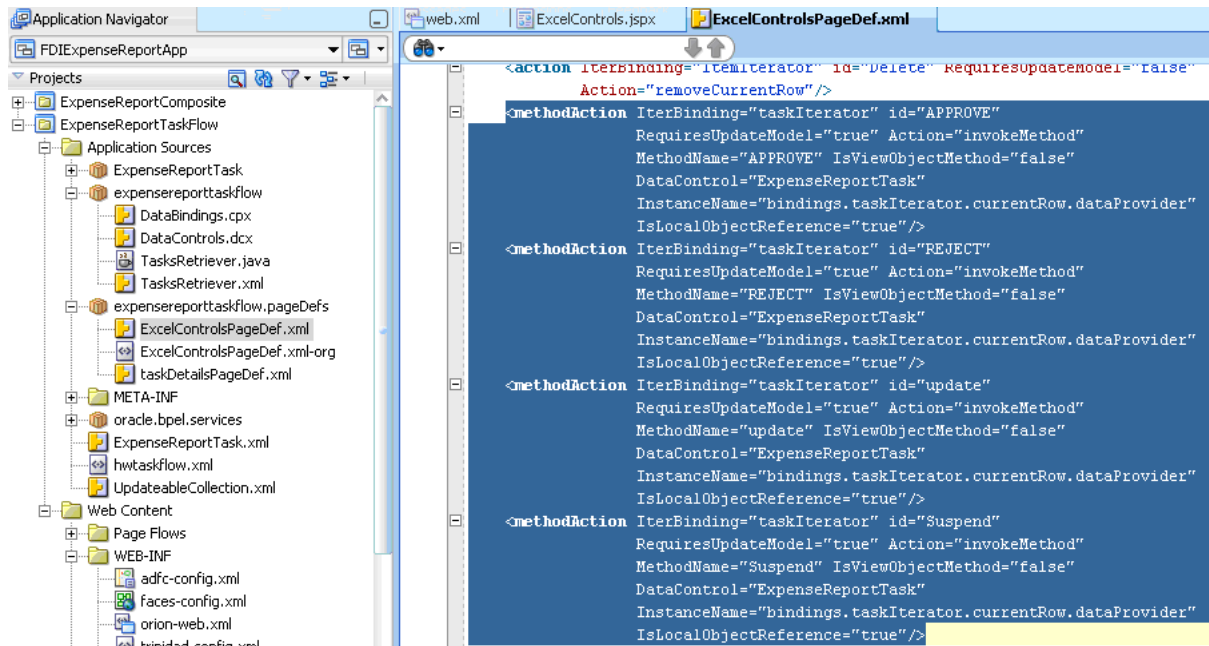
Figure 32–7 Oracle JDeveloper: Creating a JSF JSP Page



5. Drag and drop the task node from the Data Controls palette to `ExcelControls.jspx`. Select **Human Task**, then select **Complete task with payload**. [Figure 32–8](#) illustrates the sequence of menus you use. Click **OK** on dialogs that pop up.

Figure 32–8 Oracle JDeveloper: Creating an ADF Read-Only Form

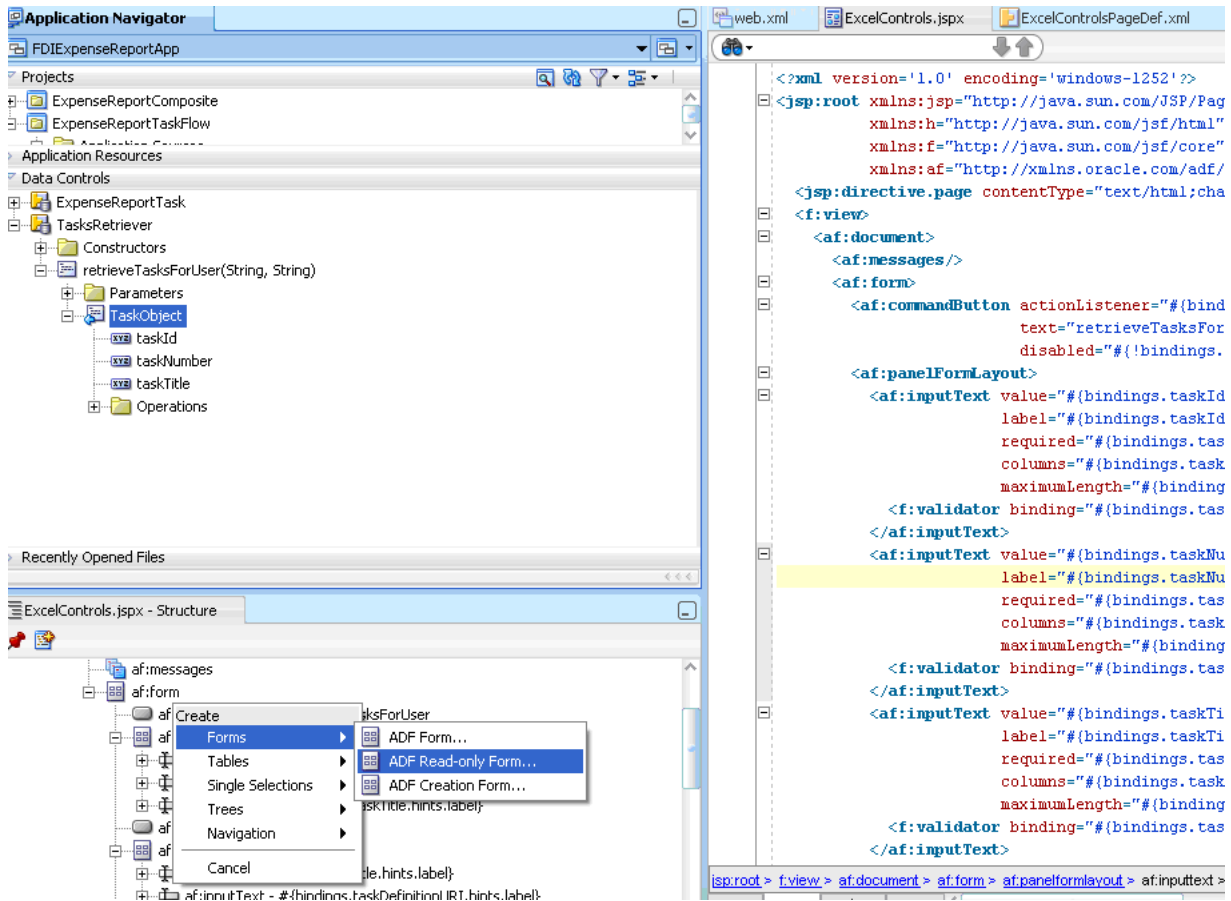
6. Drag and drop one or more task actions to the .jspx file. In this example, as illustrated in Figure 32–9, the actions **APPROVE**, **REJECT**, **update**, and **Suspend** are added to create the entries in the page definition.

Figure 32–9 Oracle JDeveloper: Configuring the Page Definition File

7. Drag and drop the **retrieveTasksForUser()** method from the Data Controls palette (expand the node **TasksRetriever**) to **ExcelControls.jspx**. For now, click **OK** on the Edit Action Binding dialog. This creates a binding in **ExcelControlsPageDef.xml** to extract all assigned tasks for the logged-in user.

8. Drag and drop **TaskObject** from the Data Control palette to **ExcelControls.jspx** to create an ADF read-only form. Verify that a corresponding `<methodIterator>` executable and `<attributeValues>` bindings are created in `ExcelControlsPageDef.xml`. [Figure 32–10](#) provides details.

Figure 32–10 Oracle JDeveloper: Page Definition File



9. Depending on the number of task details to be exposed in the Excel workbook, drag and drop as many ADF controls as needed. In this example, you expose only as many task details as needed to develop a minimally-operational workbook.
10. Create a list binding in `ExcelControlsPageDef.xml` that can create a list of assigned tasks in the Excel workbook. Add the following entry to the `<bindings>` element in the page definition.

```
<list ListOperMode="navigation"
      IterBinding="retrieveTasksForUserIterator" id="retrievedTaskList"
      StaticList="false">
  <AttrNames>
    <Item Value="taskNumber"/>
  </AttrNames>
</list>
```

11. Similarly add the following list binding in `ExcelControlsPageDef.xml` that can be later used to create a list of an updatable table of expense items in the Excel workbook.

```
<list ListOperMode="navigation" IterBinding="ItemIterator"
      id="expenseItemsList" StaticList="false">
```

```

    <AttrNames>
      <Item Value="itemName"/>
    </AttrNames>
  </list>

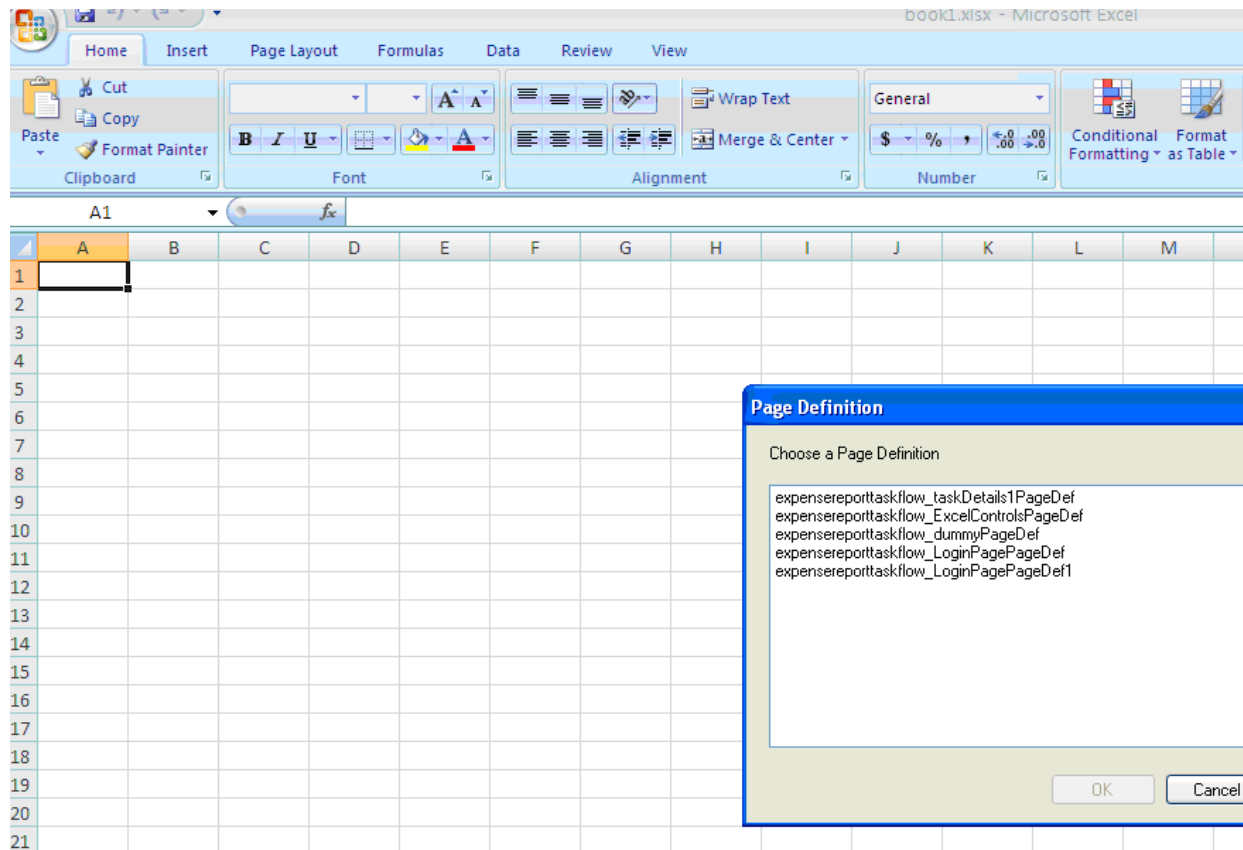
```

12. Click **Save All**. Right-click the project and click **Rebuild**. Make sure that there are no compilation errors and the build completes successfully.

32.2.3.4 Task 4: Prepare the Excel Workbook

To author the Excel workbook, follow these steps:

1. For information about desktop requirements for running the ADF-DI solution, read Section 3.1 of *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.
2. Configure security for Excel:
 - a. Open Excel.
 - b. Click the **Microsoft Office** button, then click **Excel Options**.
 - c. Click the **Trust Center** tab, then click **Trust Center Settings**.
 - d. Click the **Macro Settings** tab, then click the checkbox labeled **Trust Access** to the VBA project object model.
 - e. Click **OK**.
3. Run the setup tool that comes with the Oracle ADF-DI module. The setup tool is stored in the following folder: `JDEV_HOME\jdeveloper\adfdi\bin\excel\client`.
4. Create a new Excel workbook in the directory `project_home\public_html`. Click **View**, then click **Refresh**. This displays the Excel workbook in Oracle JDeveloper.
5. Run the conversion command on the Excel workbook. The Oracle ADF-DI module stores the conversion tool, `convert-adfdi-excel-solution.exe`, in `ORACLE_JDEVELOPER_HOME\jdeveloper\adfdi\bin\excel\convert`. To convert the Excel workbook, execute the following command:
`convert-adfdi-excel-solution.exe <workbook.xlsx> -attach`
 The Excel workbook is now enabled to use the Oracle ADF-DI framework.
6. Open the Excel workbook and choose a page definition. In this use case, the page definition is `expensereporttaskflow_ExcelControlsPageDef`. [Figure 32–11](#) provides details.

Figure 32–11 Excel: Page Definition Dialog

7. In the Document Actions pane, select **Workbook Properties**.
8. Specify ProtectedWebPage: `http://application_server:port/workflow/application_name/faces/app/logon`. (Note that this URL is protected and triggers form authentication. See [Section 32.2.3.2, "Task 2: Set up Authentication."](#))

Specify WebAppRoot: `http://application_server:port/workflow/application_name`. Click **OK**.

[Figure 32–12](#) provides details.

Figure 32–12 Excel: Setting WebAppRoot

Edit Workbook Properties

Edit the properties and press OK to save your changes.

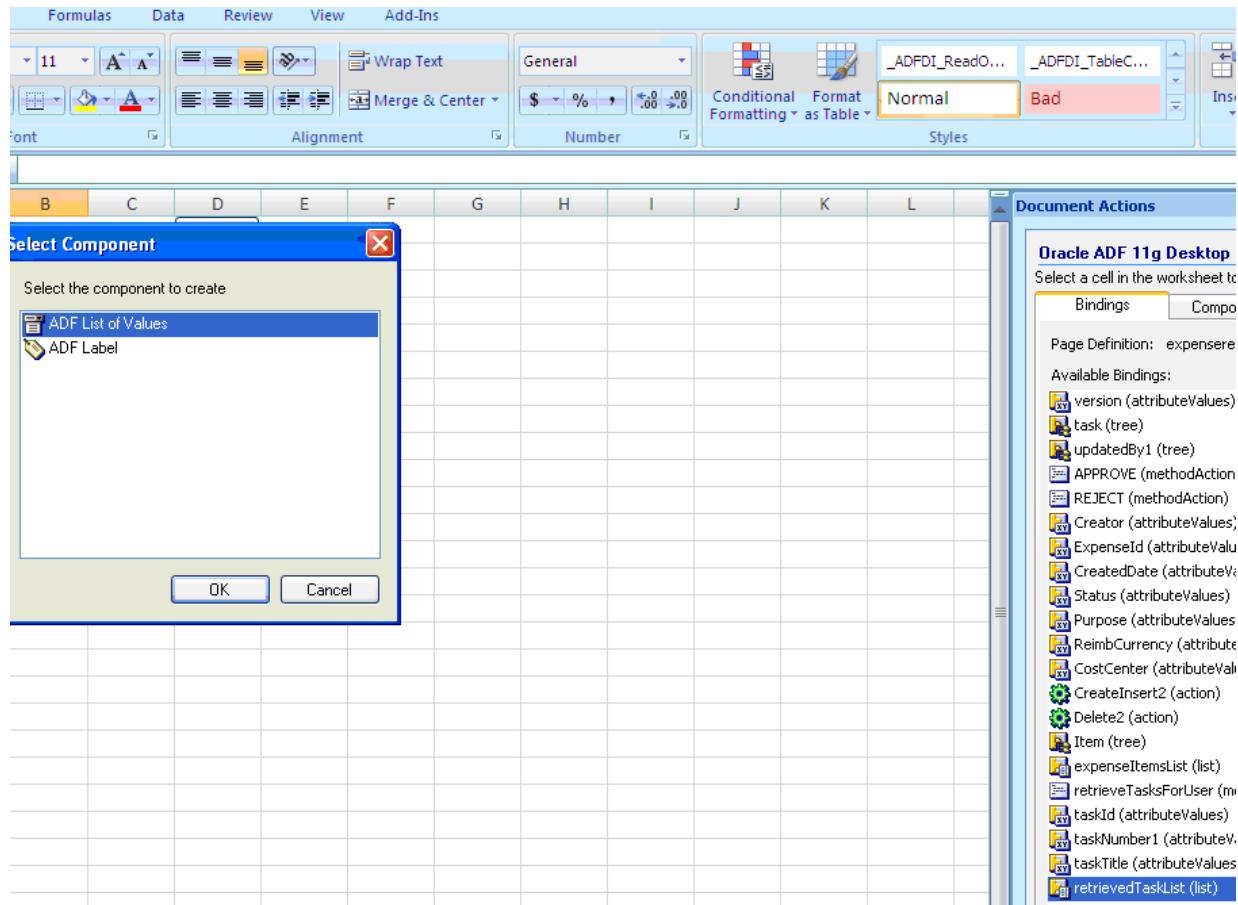
Behavior	
ToolBarMenu	
Data	
BrandingItems	NameValuePair[] Array
RemoteServletPath	/adfdiRemoteServlet
Resources	
WebAppRoot	http://stanf01.us.oracle.com:/workflow/
Design	
Annotation	
ApplicationHomeFolder	D:\Data\jdevProjects\OfficeDemo\FDIE\
Project	ExpenseReportTaskFlow
WebPagesFolder	
WorkbookID	10ba9825-9018-431f-83e9-7021f093464c
Security	
Login	
ProtectedWebPage	/faces/ExcelControls.jspx
WindowSize	600,625

WebAppRoot
The root URL for the target web application.

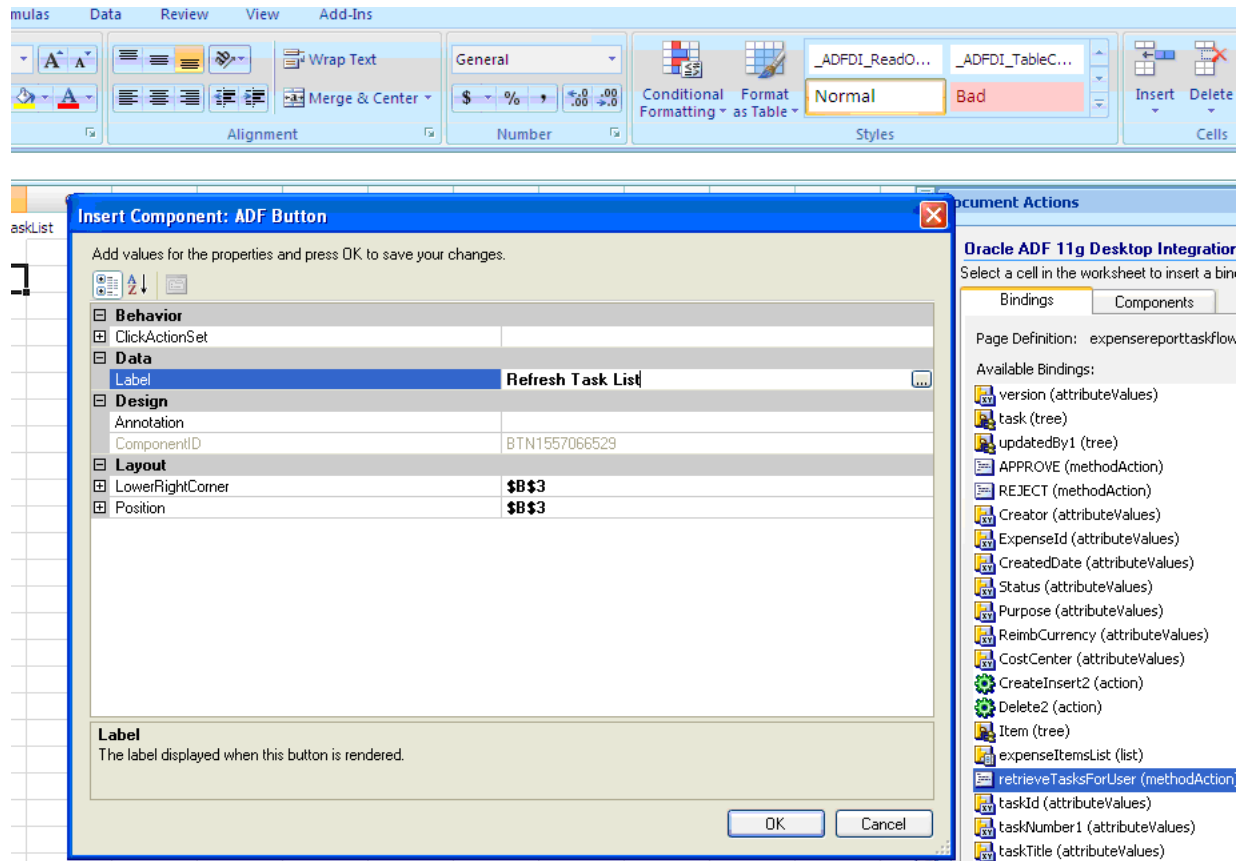
OK

For more information, see *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

- From the Document Actions pane, insert ADF Bindings to create the corresponding fields in the Excel workbook. For further details on specific components, see *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*. For instance, insert binding **retrievedTaskList** to create a list of values. [Figure 32–13](#) provides details.

Figure 32–13 Excel: Creating a List of Values

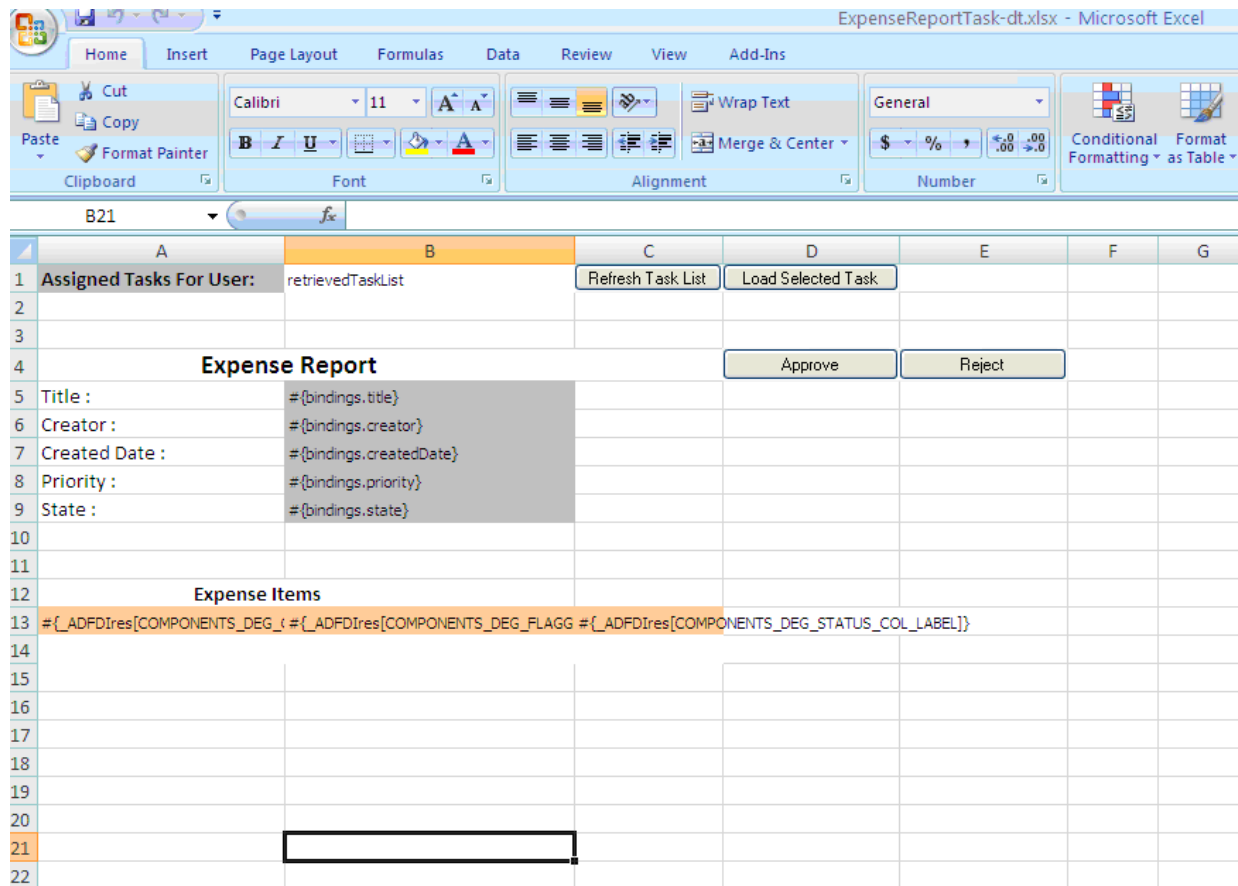
10. Insert a **methodAction** binding to create a button in Excel. [Figure 32–14](#) provides details.

Figure 32–14 Excel: Inserting a methodAction Binding

11. Insert a tree binding to create an ADF Table component. A Table component is an updatable table of records in Excel. For instance, the list binding `expenseItemsList` is a candidate for a Table component.

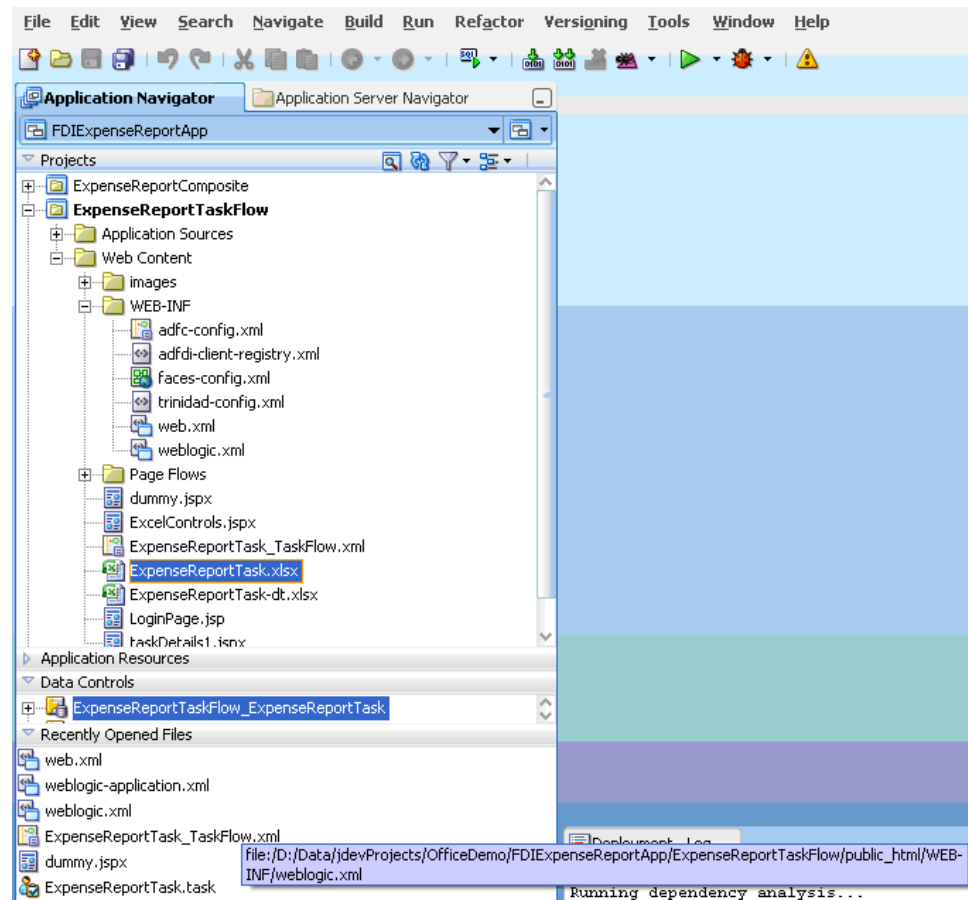
See Also: *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework* for further information about creating and modifying a Table component.

A completed Excel workbook for an expense report application looks similar to that shown in [Figure 32–15](#).

Figure 32–15 Excel Workbook Integrated with Oracle ADF-DI

12. Publish the workbook by following these steps:

- a. On the toolbar, click **Publish**. The Publish Workbook dialog appears.
 - b. In the **File name** field, specify the name as *workflow_name.xls*. The workflow name is the value of the element `WorkflowName` specified in *project_home\adfmsrc\hwtaskflow.xml*. In this example, the name of the published Excel workbook is *ExpenseReportTask.xls*.
13. In Oracle JDeveloper, click **View**, then click **Refresh**. Verify that the published workbook is visible under **Web Content**, as illustrated in [Figure 32–16](#).

Figure 32–16 Oracle JDeveloper: Verifying Workbook Under WebContent

14. Click **Save All**. The ADF task flow is now ready for deployment.

32.2.3.5 Task 5: Deploy the ADF Task Flow

To deploy the ADF task flow, follow these steps:

1. For the Excel workbook to be sent as an attachment when a task is assigned, you must configure the corresponding task in the SCA Composite:
 - a. In Oracle JDeveloper, open the SCA composite project that corresponds to the ADF task flow.
 - b. Open the `.task` file.
 - c. In the **Advanced** tab of the **Notification** section of the Human Task Editor, verify that **Send task attachments with email notifications** checkbox is checked.
2. Deploy the application. To perform a deployment, right-click the SOA Composite, select **Deploy**, select the project name, and follow the pages on the deployment wizard.
3. Deploy the ADF task flow. In the Application Navigator, expand **Projects**, and select the application. Then select **Deploy**, then **application_TaskFlow** (In this example, the application task flow is **ExpenseReportTaskFlow**), then follow the pages on the deployment wizard.

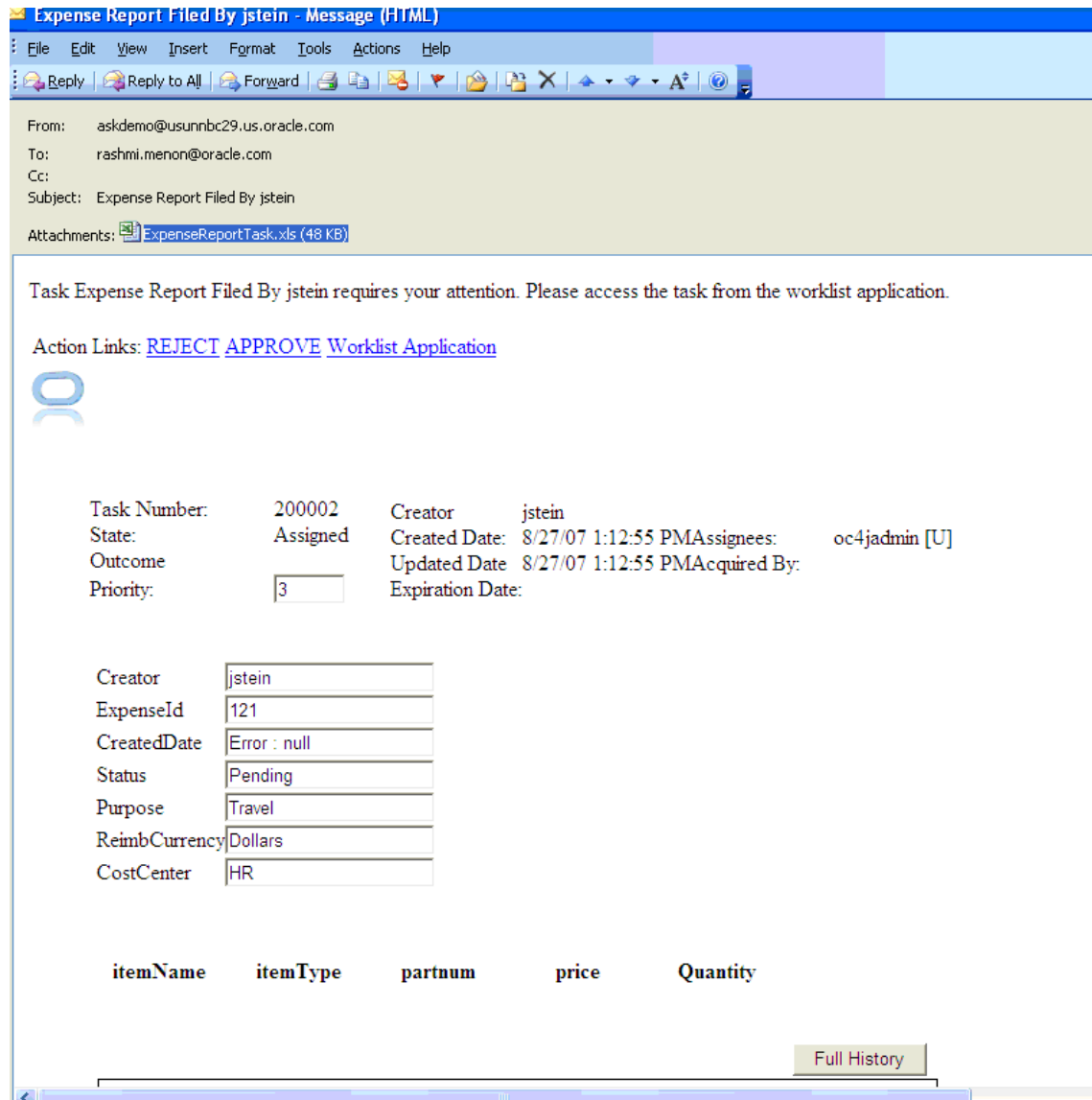
At this point, the ADF task flow is successfully deployed.

32.2.3.6 Task 6: Test the Deployed Application

To test the deployed application, follow these steps:

1. Invoke the deployed SOA composite and verify that the assignee receives the Excel workbook as part of the email notification. [Figure 32–17](#) provides details.

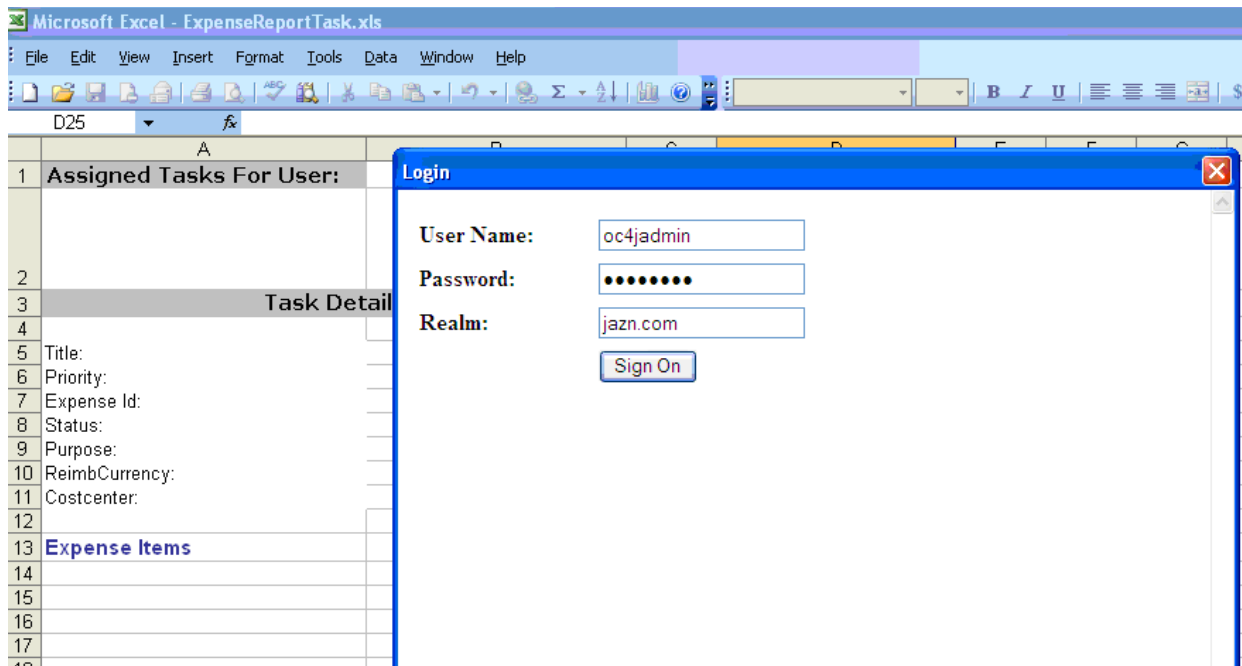
Figure 32–17 *Excel Workbook Attached to an Email*



Note: To successfully open and execute the workbook, the user's desktop host should have the correct security policy and must run the `caspol` command to grant trust to the client assemblies hosted on the network share.

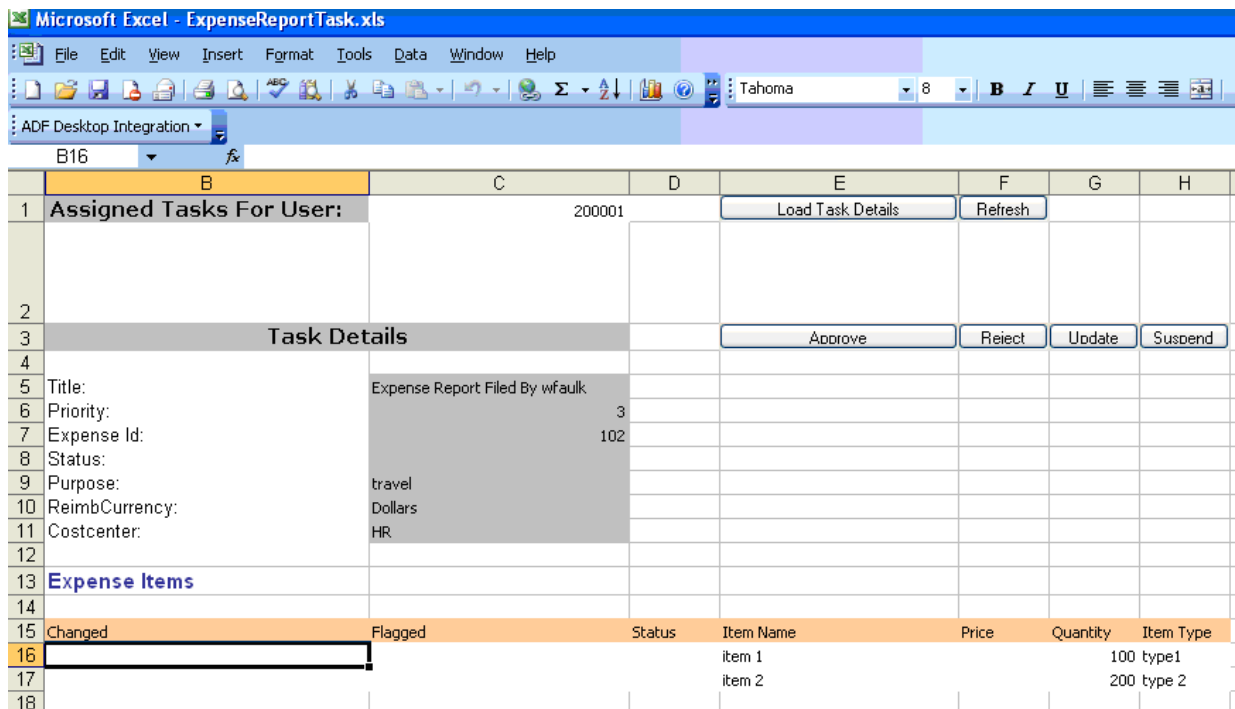
2. Open the Excel workbook. You are directed to a login page (This is `LoginPage.jsp` from [Section 32.1.2, "How to Create a Dummy JSF Page."](#)) Enter your security credentials. [Figure 32–18](#) provides details.

Figure 32–18 Desktop-Integrated Excel Workbook: Login Page



3. Examine the workbook to verify the following:
 - All the assigned tasks for the logged-in user are retrieved in the Excel workbook. Figure 32–19 provides details.

Figure 32–19 ADF Desktop-Integrated Excel Workbook with Assigned Tasks



- You can navigate to the needed task from the list of assigned tasks and update it as required. For instance, as illustrated in [Figure 32–20](#), in the Expense Report application, you can upload new expense items.

Figure 32–20 ADF Desktop-Integrated Excel Workbook Uploading New Items

Changed	Flagged	Status	Item Name	Price	Quantity
			Item 1		1
			Item 2		2
			Item 3		3
			NEW Item		

- The Status column in the workbook indicates if the upload was successful. [Figure 32–21](#) provides details.

Figure 32–21 ADF Desktop-Integrated Excel Workbook

Changed	Flagged	Status	Item Name	Price	Quantity
			Item 1		1
			Item 2		2
			Item 3		3
			NEW Item		

Also, you can perform actions on the task by clicking **Approve**, **Reject**, **Update**, or **Suspend**. [Figure 32-22](#) provides details.

Figure 32–22 Task Actions

[illegible]

Configuring Task List Portlets

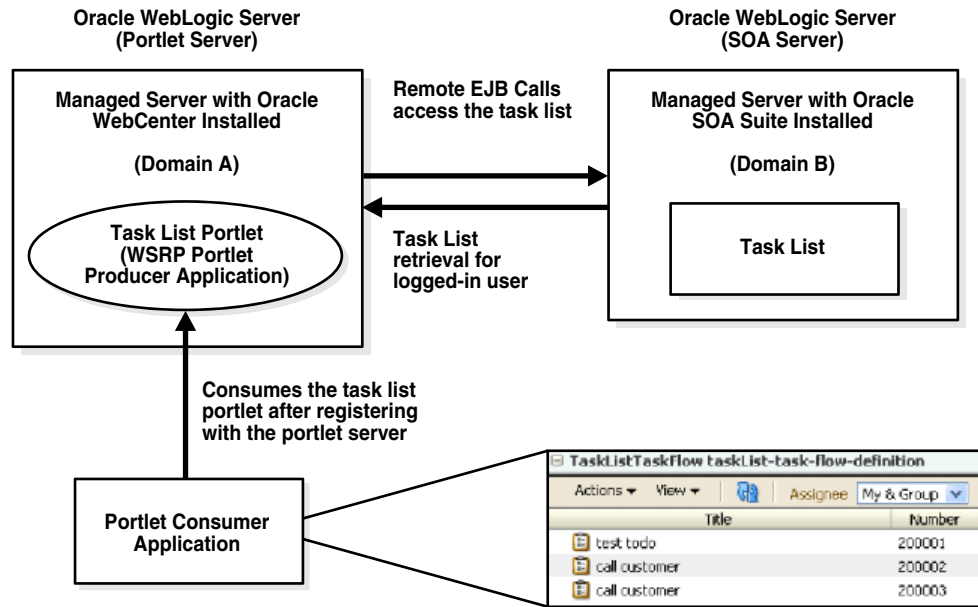
This chapter describes how to configure the task list portlets. This action enables you to review and act upon worklist tasks from an Oracle WebCenter portlet.

This chapter includes the following sections:

- [Section 33.1, "Introduction to Task List Portlets"](#)
- [Section 33.2, "Deploying the Task List Portlet Producer Application to a Portlet Server"](#)
- [Section 33.3, "Creating a Portlet Consumer Application for Embedding the Task List Portlet"](#)
- [Section 33.4, "Passing Worklist Portlet Parameters"](#)

33.1 Introduction to Task List Portlets

The worklist task list is exposed as a JSR-168 Web Services for Remote Portlets (WSRP) portlet and can be embedded in portal applications. This portlet enables you to check the business and personal ToDo tasks assigned to the user and take actions on the tasks. You build a consumer application that can consume the JSR-168 portlet hosted by the task list portlet producer application. Any consumer can consume the portlet after registering with the portlet producer (the Oracle WebLogic Server portlet server). The portlet also supports many customizations through parameters, which are described in [Section 33.4, "Passing Worklist Portlet Parameters."](#) [Figure 33–1](#) shows the high level portlet deployment and usage.

Figure 33–1 High Level Portlet Deployment and Usage

33.2 Deploying the Task List Portlet Producer Application to a Portlet Server

This section describes how to deploy and configure the task list portlet producer application on a managed portlet server.

33.2.1 Deployment Prerequisites

This section describes deployment prerequisites for the task list portlet producer application.

- Since the task list portlet is a WSRP portlet producer application, it must be deployed on a managed server configured for deploying portlet producer applications. For this to occur, you must install Oracle WebCenter.
- Oracle WebCenter and Oracle SOA Suite must be installed in different domains.
- If the task list portlet producer application is installed on the SOA server, you can skip the steps described in [Section 33.2.3, "How to Connect the Task List Producer to the Remote SOA Server."](#)
- The task list portlet producer application is deployed on the Oracle WebLogic Server portlet server shown in [Figure 33–1](#) (the host on which Oracle WebCenter is installed). The portlet server contacts the remote Oracle WebLogic Server SOA server to access the task list using remote Enterprise JavaBeans (EJB) calls. The portlet producer application EAR file is provided on the SOA server in the following directory:

`Oracle_Home/SOA_Home/soa/applications`

(for example, `/fmwhome/AS11gR1SOA/soa/applications`)

- The shared library `oracle.soa.workflow.wc` must be targeted to the Oracle WebLogic Server portlet managed server. See [Section 33.2.2, "How to Deploy the Task List Portlet Producer Application"](#) for instructions.

33.2.2 How to Deploy the Task List Portlet Producer Application

To deploy the task list portlet producer application:

1. Install Oracle WebCenter as described in *Oracle Fusion Middleware Installation Guide for Oracle WebCenter*.
2. For this administration domain, start both the Oracle WebLogic Administration Server and the Oracle WebLogic Server portlet managed server. See *Oracle Fusion Middleware Administrator's Guide* for instructions on starting administration and managed servers.
3. Because the task list portlet producer application uses the deployed library `oracle.soa.workflow.wc`, you must confirm that the library is targeted to the Oracle WebLogic Server portlet managed server.

- a. Log in to Oracle WebLogic Server Administration Console.

`http://hostname:port/console`

where *hostname* and *port* are the hostname and port for the Oracle WebLogic Server Administration Console.

- b. Go to **Deployments** > **oracle.soa.workflow.wc** > **Targets**.
 - c. See if **WLS_Portlet** is checked. If not, check it and save your updates.
4. Deploy the **TaskListTaskFlow.ear** file on the Oracle WebLogic Server portlet managed server.
 - a. In the **Domain Structure** section, click **Deployments**.
 - b. In the **Deployment** section, click **Install**.
 - c. Navigate to and select to install **TaskListTaskFlow.ear** as an application. For example:

`/Oracle_Home/SOA_Home/soa/applications/TaskListTaskFlow.ear`

5. Ensure that the WSRP producer application is running by accessing the WSDL from a web browser:

`http://server:port/TaskListTaskFlow/portlets/wsrp2?WSDL`

33.2.3 How to Connect the Task List Producer to the Remote SOA Server

The task list portlet producer application communicates with the remote Oracle WebLogic Server SOA managed server to get the task list for the logged-in user. See [Figure 33–1](#) for details. The task list portlet producer application uses remote EJB calls to the human workflow services API to achieve this. Therefore, you must configure the remote JNDI providers on the Oracle WebLogic Server on which Oracle WebCenter is installed.

33.2.3.1 How to Define the Foreign JNDI on the Oracle WebCenter Oracle WebLogic Server

To define the foreign JNDI on the Oracle WebCenter Oracle WebLogic Server:

1. Log in to Oracle WebLogic Server Administration Console:

`http://remote_hostname:remote_port/console`

where *remote_hostname* and *remote_port* are the hostname and port for the remote Oracle WebCenter Oracle WebLogic Server.

2. Navigate to **Domain Structure > Services > Foreign JNDI Providers**.
3. Click **New**.
4. In the **Name** field, enter `ForeignJNDIProvider-SOA`.
5. Click **OK**.
6. Click the **ForeignJNDIProvider-SOA** link.

The Settings for ForeignJNDIProvider-SOA page appears.

7. Enter values for the fields listed in [Table 33–1](#), then click **Save**.

Table 33–1 Parameters and Values

Field	Description
Initial Context Factory	Enter <code>weblogic.jndi.WLInitialContextFactory</code> .
Provider URL	Enter <code>t3://soa_hostname:soa_port/soa-infra</code> . Note: Replace <i>soa_hostname</i> and <i>soa_port</i> with the hostname and port for the remote Oracle WebLogic Server SOA server that includes the task list to retrieve.
User	Enter <code>weblogic</code> .
Password	Enter the password for the user.
Confirm Password	Enter the same password again.

8. Click **ForeignJNDIProvider-SOA**.
9. Click the **Links** tab.
10. Under **Foreign JNDI Links**, click **New**.

The Create a Foreign JNDI Link page appears.

11. Enter values for the fields listed in [Table 33–2](#), and click **OK**.

Table 33–2 Parameters and Values

Field	Values
Name	Enter <code>RuntimeConfigService</code> .
Local JNDI Name	Enter <code>RuntimeConfigService</code> .
Remote JNDI Name	Enter <code>RuntimeConfigService</code> .

12. Repeat Step 11 six times and enter the values shown in [Table 33–3](#) for the **Name**, **Local JNDI Name**, and **Remote JNDI Name** fields.

Table 33–3 Parameters and Values

The...	Enter This Value in the Name, Local JNDI Name, and Remote JNDI Name Fields, and click OK...
First time	<code>ejb/bpel/services/workflow/TaskServiceBean</code>
Second time	<code>ejb/bpel/services/workflow/TaskMetadataServiceBean</code>
Third time	<code>TaskReportServiceBean</code>
Fourth time	<code>TaskEvidenceServiceBean</code>

Table 33–3 (Cont.) Parameters and Values

The...	Enter This Value in the Name, Local JNDI Name, and Remote JNDI Name Fields, and click OK...
Fifth time	TaskQueryService
Sixth time	UserMetadataService

For more information about configuring a foreign JNDI provider, see the *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.

33.2.3.2 How to Configure EJB Identity Propagation

The task list portlet producer application must be configured so that the already-authenticated user token in the consumer application is passed to the producer-managed server and then to the remote SOA server. This can be achieved by enabling global trust between the concerned domains. For more information about enabling cross domain security between Oracle WebLogic Server domains, see *Oracle Fusion Middleware Securing Oracle WebLogic Server*.

To configure EJB identity propagation:

1. To enable the global trust, log in to the Oracle WebLogic Server Administration Console of the Oracle WebCenter Oracle WebLogic Server.
2. On the left side of the page, select the domain name that you specified during installation (for example, **soainfra**).
3. Select **Security**, and expand the **Advanced** section.
4. Modify the domain credentials.
5. Log in to the Oracle WebLogic Server Administration Console of the SOA server Oracle WebLogic Server.
6. Modify the domain credentials of the SOA server and enter the same password as entered for the Oracle WebCenter server in Step 4.
7. Click **Save**.

33.2.3.3 How to Configure the Identity Store

You must configure the authenticator of the Oracle WebCenter Oracle WebLogic Server domain to point to the same identity provider used by the SOA server.

Note that either the user name used to log in to the consumer application must be present in the identity stores of the portlet server and SOA server or all three servers must point to the same identity store. The three impacted servers are as follows:

- The Oracle SOA Suite managed server
- The Oracle WebCenter managed server on which the task list portlet producer application is deployed
- The server on which the portlet consumer application is deployed

The user first logs in to the consumer application. Therefore, the user must be present in the identity store of this server. Then, when the consumer application contacts the task list portlet producer application, it must propagate the user name to the Oracle WebCenter managed server. The same user name must also be present in the identity store of this server. Then, to fetch the Oracle SOA Suite data, the task list portlet producer application contacts the Oracle SOA Suite managed server. Therefore, it must again propagate the user name to the SOA server. Again, the same user name must be

present in the identity store of the Oracle SOA Suite server. Alternatively, all the above servers can point to the same identity store.

To configure the identity store:

1. Log in to the Oracle WebLogic Server Administration Console of the Oracle WebCenter Oracle WebLogic Server.
2. See Section "Reassociating the Identity Store with an External LDAP" of *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter* for instructions on configuring the identity store.
3. Follow these instructions for all three servers.

33.2.4 How to Secure the Task List Portlet Producer Application Using Web Services Security

You must perform the following tasks to secure the task list portlet producer application:

- Enable WS-Security for the task list portlet producer application
- Set up the certificate keystores

Note: Ensure that you copy the `producer.jks` file to a location in your file system that is running the task list portlet producer application. For the following example, the keystore is copied under `domain_home/config/fmwconfig`.

To secure the task list portlet producer application using web services security:

1. See Sections "Securing a WSRP Producer with WS-Security" and "Securing Oracle WebLogic Communication Services (OWLCS) with WS-Security" of *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter* for instructions on enabling WS-Security and setting up the certificate keystores.

While following the instructions in those sections, you access the following pages in Oracle Enterprise Manager Fusion Middleware Control.

- a. In the navigator on the left side, select **Farm_base_domain > WebLogic Domain**.
where *base_domain* is the domain name for this example.
- b. Right-click *base_domain* and select **Security > Security Provider Configuration**.
- c. Access the **Keystore** section at the bottom of the provider configuration page and click **Configure**, as shown in [Figure 33-2](#).

Figure 33–2 Keystore Section**Web Services Manager Authentication Providers**

You can configure the login modules and keystore for Web Services Manager authentication.

Login Modules

The following table lists all configured login modules for Web Services Manager. Use this list to create, configure or delete a login module.

					
Name	Class	Control Flag	Description		
saml.loginmodule	oracle.security.jps.internal.jaas.module.saml.JpsSAMLLoginModule	Required	SAML L		
krb5.loginmodule	com.sun.security.auth.module.Krb5LoginModule	Required	Kerber		
digest.authenticator.log	oracle.security.jps.internal.jaas.module.digest.DigestLoginModule	Required	Digest		
certificate.authenticator	oracle.security.jps.internal.jaas.module.x509.X509LoginModule	Required	X509 C		
wss.digest.loginmodule	oracle.security.jps.internal.jaas.module.digest.WSSDigestLoginMc	Required	WSS D		
user.authentication.logir	oracle.security.jps.internal.jaas.module.authentication.JpsUserAu	Required	User A		
user.assertion.loginmodi	oracle.security.jps.internal.jaas.module.assertion.JpsUserAsserti	Required	User A		

Keystore

Use this section to specify the keystore used to store public and private keys for all secure connections within the WebLogic Domain.



- d. Enter details for keystore management and identity certificates, as shown in [Figure 33–3](#). Section "Securing a WSRP Producer with WS-Security" of *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter* provides specific details.

Figure 33–3 Keystore Configuration**Keystore Configuration**

A keystore is a key database that contains both public and private keys. Keystore needs to be configured only at the WebLogic Domain level. You will need to provide the keystore name, path, password and information about default identity certificates.

☒ **Configure Keystore Management**

 **TIP** To remove keystore configuration, uncheck Configure Keystore Management checkbox above and click OK.

Keystore Type **JKS**

* Keystore Path

* Password

* Confirm Password

Identity Certificates

Specify the default identity certificates (signature and encryption keys) for this keystore. Web Services that are configured to use this keystore will use these identity certificates.

Signature Key

* Key Alias

* Signature Password

* Confirm Password

Encryption Key

* Crypt Alias

* Crypt Password

* Confirm Password

- e. When complete, click **OK**.
- f. Restart the managed portlet server and the administration server for the managed portlet server.

33.2.5 How to Specify the Inbound Security Policy

You now specify the inbound security policy. This section assumes that the keystore configuration steps described in [Section 33.2.4](#), "How to Secure the Task List Portlet Producer Application Using Web Services Security" have been completed.

To specify the inbound security policy:

1. In Oracle Enterprise Manager Fusion Middleware Control under **Application Deployments**, navigate to the portlet producer application node.
2. Click **Application Deployments > TaskListTaskFlow (WLS_Portlet)**.
3. Select **menu > Application Deployments > Web Services**.
4. Select the markup port from the page that is displayed, as shown in [Figure 33–4](#).

Figure 33–4 Markup Port Selection

Name	Endpoint Enabled	Invocations Completed
WSRP_v2_Service		
WSRP_v2_Markup_Service	Enabled	0
WSRP_v2_PortletManagement_...	Enabled	0
WSRP_v2_ServiceDescription_S...	Enabled	0
WSRP_v2_Registration_Service	Enabled	0
WSRP_v1_Service		
WSRPServiceDescriptionService	Enabled	0
WSRPRegistrationService	Enabled	0
WSRPBaseService	Enabled	0
WSRPPortletManagementService	Enabled	0

5. On the page that is displayed, click the **Policies** tab.
6. Click the **Attach/Detach** button.
7. Attach and detach policies appropriate to your use of the task list portlets producer application, as shown in [Figure 33–5](#).

Figure 33–5 Policy Attachment and Detachment

Attached Policies			
Policy Name	Category	Enabled	Description
oracle/wss10_username_token_with_message_protection_service_policy	Security	Yes	This policy enforces message

Policy Name	Category	Enabled	Description
oracle/wsaddr_policy	WS-Addres...	Yes	This policy causes the platfor
oracle/log_policy	Management	Yes	This policy causes the reques
oracle/wsmtom_policy	MTOM Atta...	Yes	This Message Transmission O
oracle/binding_authorization_denyall_policy	Security	Yes	This policy is a special case of

8. Once complete, click **OK** in each open page.
9. Restart the managed server to which the task list portlet producer application is deployed.

33.3 Creating a Portlet Consumer Application for Embedding the Task List Portlet

You now create a portlet consumer application for embedding the task list portlet, as shown in [Figure 33–1](#).

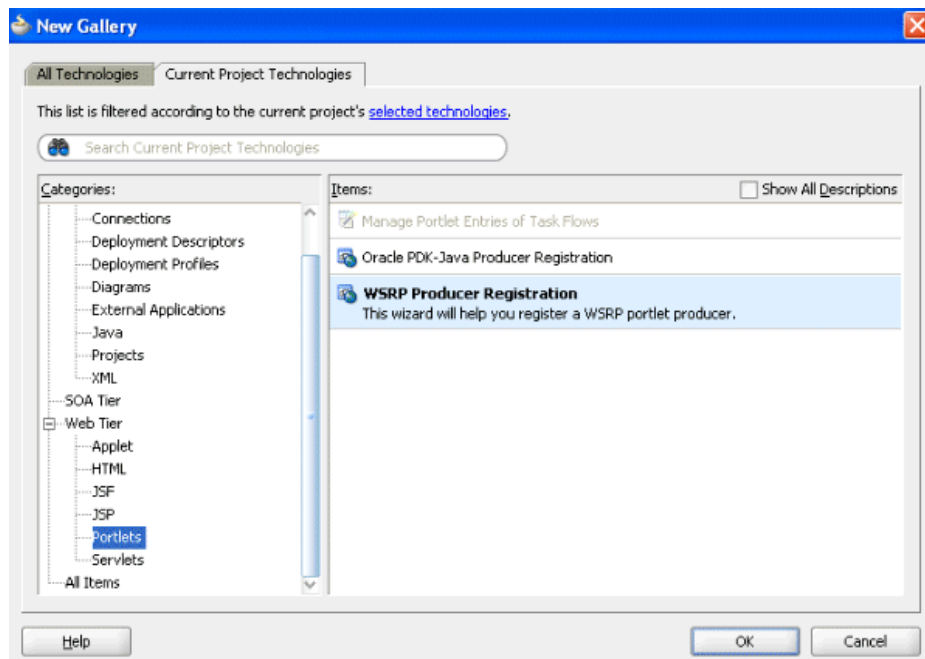
Ensure that you have already deployed and configured the task list portlet producer application as described in [Section 33.2, "Deploying the Task List Portlet Producer Application to a Portlet Server"](#) and verified that it is running. Note that the portlet consumer application can only be deployed on a managed server that has Oracle WebCenter installed.

33.3.1 How To Create a Portlet Consumer Application for Embedding the Task List Portlet

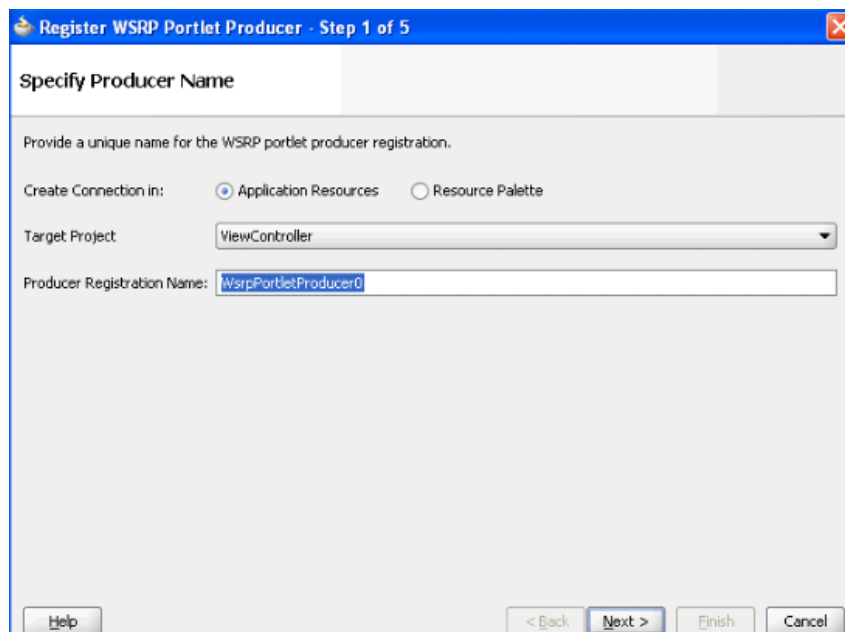
Follow these procedures to create a consumer application for embedding the task list portlet.

To create a portlet consumer application for embedding the task list portlet:

1. Create a new Oracle WebCenter application in Oracle JDeveloper:
 - a. From the **File** main menu, select **New > Application**.
 - b. Select **WebCenter Application**, and click **OK**.
 - c. In the **Application Name** field, enter a name (for this example, `TaskListConsumer` is entered).
 - d. Click **Finish**.
2. Add a new JSPX page to the application `consumer.jspx`.
3. Register the WSRP producer with the consumer by dragging and dropping the portlet on `consumer.jspx`:
 - a. In the Application Navigator, right-click **View Controller** and select **New**.
 - b. Click **Portlets** under **web tier**.
 - c. Select **WSRP Producer Registration** in the right hand pane, as shown in [Figure 33–6](#).

Figure 33–6 WSRP Producer Registration

- d. Click **OK**.
A Register WSRP Portlet Producer wizard is displayed.
- e. Click **Next** on the Welcome page.
- f. Check the **Application Resources** button.
- g. Provide a producer registration name, as shown in [Figure 33–7](#).

Figure 33–7 Producer Name

- h. Click **Next**.

- i. Provide the following URL endpoint:

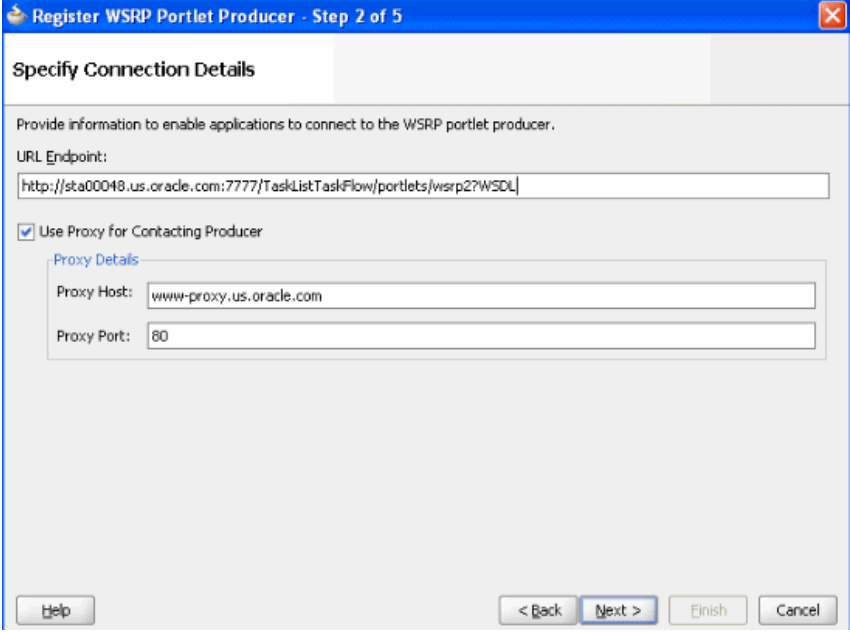
`http://server:port/TaskListTaskFlow/portlets/wsrp2?WSDL`

where *server* is the host on which the portal service is installed and *port* is the port on that server.

- j. Enter proxy details appropriate to your environment.

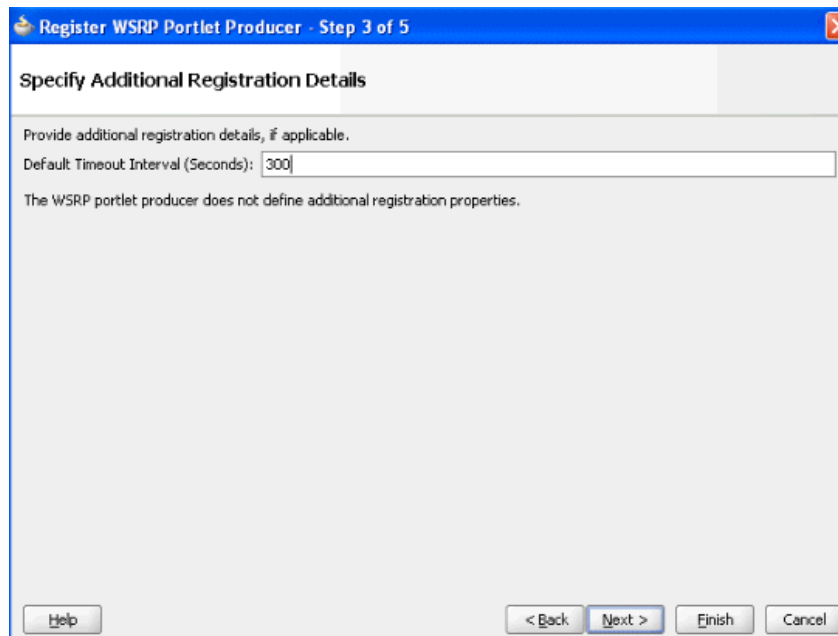
Figure 33–8 provides details.

Figure 33–8 URL Endpoint



- k. Click **Next**.

- l. Specify the execution timeout, as shown in Figure 33–9. Oracle recommends that you specify a large value, such as 300 seconds. This reduces the chance of timeout exceptions occurring during runtime.

Figure 33–9 Execution Timeout

- m. Click **Next**.

The Configure Security Attributes page appears.

- n. From the **Token Profile** list, select a token profile appropriate to your environment. For example, select the **SAML Token with Message Integrity** token profile. The token profile selected must be the same as that selected when you configured WS-Security on the task list portlet producer application, as described in Section "Securing a WSRP Producer with WS-Security" of *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter*.
- o. For the **Configuration** option, select **Custom**.
- p. Specify the default user as `fmwadmin` and the issuer name as `www.oracle.com`, as shown in [Figure 33–10](#).

Figure 33–10 Security Attribute Configuration

Register WSRP Portlet Producer - Step 4 of 5

Configure Security Attributes

If the producer supports secure identity propagation using WS-Security, configure the security attributes.

Token Profile: SAML Token with Message Integrity

Enforces message-level integrity protection and SAML-based authentication for inbound SOAP requests.

Configuration: ☐ Default ☒ Custom

Select this option to override the default Oracle Platform Security for Java configuration values with custom values.

Default User:

Issuer Name:

- q. Copy `consumer.jks` to your local directory.
 - r. Click the **Browse** button to select the consumer keystore (`consumer.jks` file) you used for configuring web service security for the producer application in [Section 33.2.4, "How to Secure the Task List Portlet Producer Application Using Web Services Security."](#)
 - s. Complete the remaining fields.
- [Figure 33–11](#) provides details.

Figure 33–11 Key Store Specification

Register WSRP Portlet Producer - Step 5 of 5

Specify Key Store

Specify the location of the Key Store that contains the private key used to sign the security assertions. Select the signature key alias that corresponds to the private key to be used for signing.

Store Path: Browse

Store Password:

Store Type:

Signature Key Alias: consumer

Signature Key Password:

Encryption Key Alias: consumer

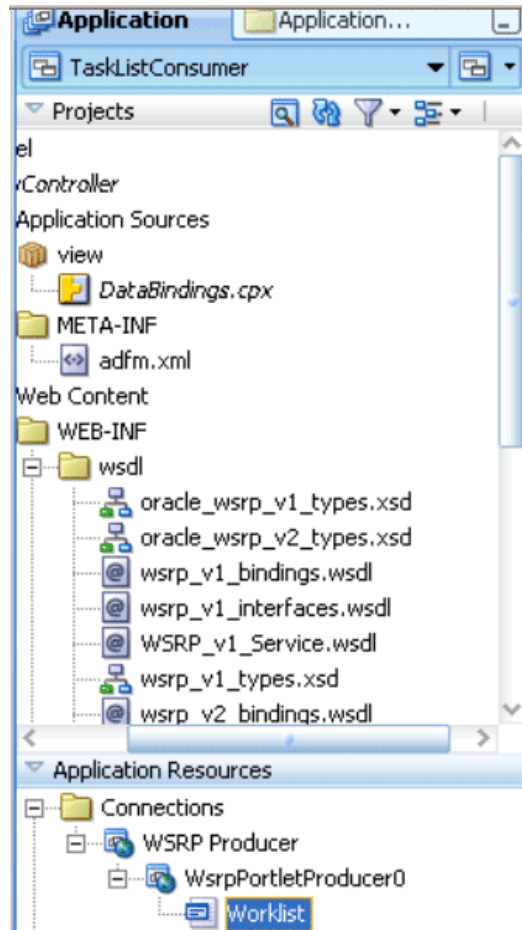
Encryption Key Password:

- t. Click **Finish**.

The registered portlets appear under **Application Resources**.

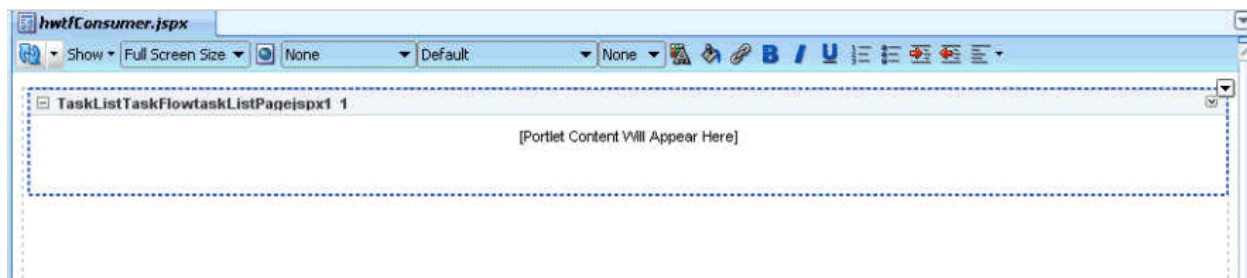
- u. Select the token profile based on the requirements of your application, as shown in [Figure 33–12](#).

Figure 33–12 Token Profile Selection

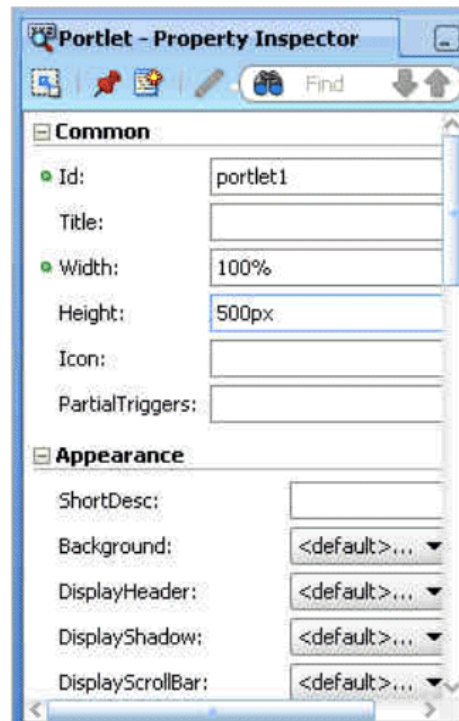


- v. Drag the task list portlet named **Worklist** onto the JSPX page `consumer.jspx`, as shown in [Figure 33–13](#).

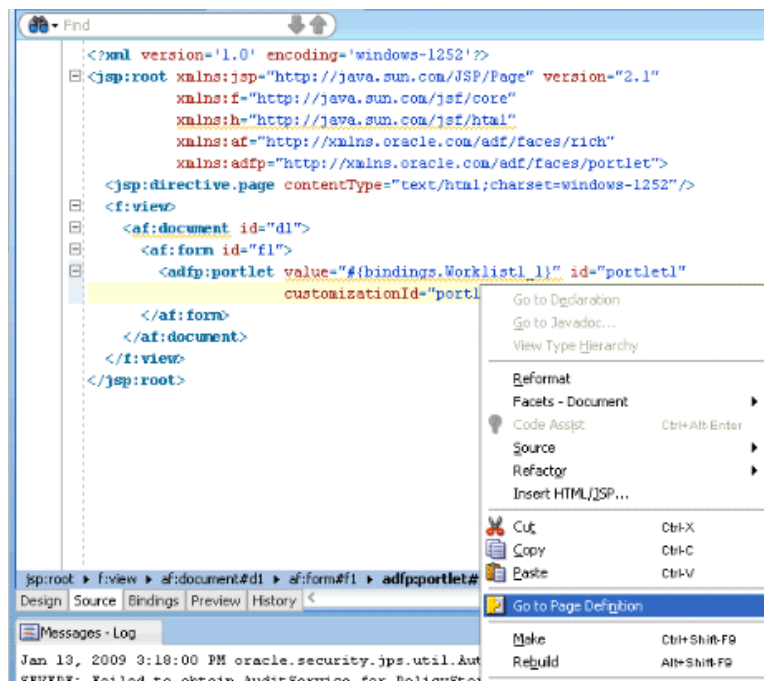
Figure 33–13 consumer.jspx



- w. Specify the height and width for the task list portlet suitable for your page, as shown in [Figure 33–14](#). This dialog typically appears at the bottom right when you select the portlet component that is dragged onto the page. If this dialog does not appear, select **Property Inspector** from the **View** main menu.

Figure 33–14 Height and Width Specifications for the Portlet

- x. Right-click **consumer.jspx** in the designer and select **Go to Page Definition**, as shown in Figure 33–15.

Figure 33–15 Page Definition Selection

This takes you to **consumerPageDef.xml**.

4. Provide values for the parameters described in [Table 33–4](#). See [Section 33.4, "Passing Worklist Portlet Parameters"](#) for additional details.

Table 33–4 Parameters and Values

Parameter	Description of Value
soaURL Used when the SOA server and the portlet server are different. The task details for the ToDo task require this URL.	<code><variable Name="Worklist1_1_soaURL" Type="java.lang.Object" DefaultValue="\${ 'http://soa_host:soa_port' }" /></code>
refreshURL The complete URL of the page, including the task list portlet.	<code><variable Name="Worklist1_1_refreshURL" Type="java.lang.Object" DefaultValue="\${ 'http://soa_host:soa_port/HWTFConsumer/faces' }" /></code>

[Figure 33–16](#) provides details.

Figure 33–16 consumerPageDef.xml

```

<executables>
  <variableIterator id="variables">
    <variable Name="Worklist1_1_showViewsPanel" Type="java.lang.Object"/>
    <variable Name="Worklist1_1_showTaskDetailsPanel"
      Type="java.lang.Object"/>
    <variable Name="Worklist1_1_wfCtxID" Type="java.lang.Object"/>
    <variable Name="Worklist1_1_soaURL" Type="java.lang.Object" DefaultValue="${ 'http://soahost:soaport' }"/>
    <variable Name="Worklist1_1_refreshURL" Type="java.lang.Object" DefaultValue="${ 'http://myhost:myport/HWTFConsumer/faces' }"/>
    <variable Name="Worklist1_1_showActionDropdown" Type="java.lang.Object"/>
  
```

5. Secure the Oracle WebCenter consumer application using ADF security by following the steps provided in chapter "Enabling ADF Security in a Fusion Web Application" of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* (section "How to Enable Oracle ADF Security").
6. Configure the identity store of the embedded Oracle WebLogic Server of Oracle JDeveloper to point to that of the SOA server. You can do this by following the steps described in [Section 33.2.3.3, "How to Configure the Identity Store."](#)
7. Run the **consumer.jspx** consumer application page:
 - a. Right-click the **consumer.jspx** page.
 - b. Select **Run**.

This starts the embedded Oracle WebLogic Server instance, deploys the consumer application, and shows the portlet in the **consumer.jspx** page.

33.4 Passing Worklist Portlet Parameters

The task list portlet can accept certain parameters in the `consumerPageDef.xml` file. The consumer application for the task list region can do the following:

- Pass some parameters to the producer application
- Control the display behavior of the embedded region

- Pass parameters to filter the task list, such as a list of task types and a task attributes value list

Table 33–5 shows the display parameters.

Table 33–5 Display Parameters

Parameters	Description	Values	Mandatory
displayColumnsList	A comma separated string of the columns to be displayed in the task list table.	Possible values: ■ title ■ number ■ priority ■ assignees ■ state ■ createdDate ■ expirationDate See Section 33.4.2, "Example of File Containing All Column Constants" for an example.	No
localeSource	Specifies whether to take language settings from the web browser or the identity settings.	Possible values: ■ identity (default) ■ browser	No
refreshURL	The complete URL of the page, including the task list portlet. This is a mandatory parameter if showTaskDetailsPanel is set to true. The task details in the task list region are shown in an inline frame. Therefore, if any action is taken on the task details page, it tries to refresh the task listing area. To do that, it refreshes the page URL in which the taskflow/portlet is contained. Since the taskflow does not know the URL of the container page, this URL must be passed as a parameter. If showTaskDetailsPanel is passed as false, this parameter is not required. You can get it by calling the getRequestURL() method on the HttpServletRequest/PortletRequest object.	Enter a value appropriate to your environment. See Section 33.4.2, "Example of File Containing All Column Constants" for an example.	Yes
showActionDropDown	Specifies whether to display the Actions list on the toolbar.	Possible values: ■ true (default) ■ false	No
showAssignmentFilter	Specifies whether to display the Assignment Filter Selection dropdown list in the toolbar.	Possible values: ■ true (default) ■ false	No
showSearchControl	Specifies whether to display the Quick Search text field.	Possible values: ■ true (default) ■ false	No

Table 33–5 (Cont.) Display Parameters

Parameters	Description	Values	Mandatory
showStatusFilter	Specifies whether to display the Task Status Filter Selection dropdown list in the toolbar.	Possible values: ■ true (default) ■ false	No
showTaskDetailsPanel	Specifies whether to display the task details panel.	Possible values: ■ true ■ false (default)	No
showViewFilter	Specifies whether to display the View selection dropdown list in the toolbar.	Possible values: ■ true (default) ■ false	No
showViewsPanel	Specifies whether to display the View selection panel.	Possible values: ■ true ■ false (default)	No
soaURL	Used where the SOA server and the portlet server are different. This is a mandatory parameter if showTaskDetailsPanel is set to true. The task details for the ToDo task require this URL. This is because the ToDo task is an internal application and does not know the URL of the SOA server when accessed from an application deployed on a remote non-SOA Oracle WebLogic Server. The format is as follows: <code>http://soa_host:soa_port</code>	Enter a value appropriate to your environment. See Section 33.4.2, "Example of File Containing All Column Constants" for an example.	Yes
sortColumn	The name of the column to use for sorting tasks by default in the region.	The default value is <code>createdDate</code> . See Section 33.4.2, "Example of File Containing All Column Constants" for an example.	No
sortOrder	Specifies whether to sort the task list in ascending or descending order.	Possible values: ■ asc ■ desc (default)	No
wfCtxID	Specifies the authenticated workflow context token.	Enter a value appropriate to your environment. See Section 33.4.2, "Example of File Containing All Column Constants" for an example.	No

Table 33–6 shows the filter parameters.

Table 33–6 Filter Parameters

Parameters	Description	Values	Mandatory
assignmentFilter	Specifies the type of assignee.	See Section 33.4.1, "Assignment Filter Constraints" for examples.	No
viewFilter	Specifies the selected view for which the tasks are displayed.	Enter a custom value that you create or accept the default value of Inbox.	No
taskTypesFilterList	A comma-separated list of task type values to display tasks of only the passed-in task types.	Enter a value appropriate to your environment.	No
attributesFilterOperator	The join criterion (And/Or) used for searching the specified filter criteria.	Possible values: ■ and ■ or (default)	No
attributesFilterList	The specified comma-separated list of name-value pairs used to filter tasks based on attribute values (the name is the task column name and the value is the column value).	See Section 33.4.2, "Example of File Containing All Column Constants" for an example.	No

For example, if you want to see the task with attribute filter values as `priority = 1`, `status = ASSIGNED`, and promoted mapped attribute `textAttribute1 = NorthAmerica`, then you set the values as follows:

```
attributeFilterList: priority=1, status=ASSIGNED, textAttribute1=NorthAmerica
```

and set the attribute filter operator as:

```
attributeFilterOperator: and
```

The parameters in [Table 33–5](#) and [Table 33–6](#) are defined in the page definition of the test JSPX page. [Example 33–1](#) shows the `consumerPageDef.xml` page definition file syntax when the task list is consumed as a taskflow. The attribute value has the value of the parameter.

Example 33–1 Parameter Definition

```
<parameters>
  <parameter id="showViewsPanel" value="#{testBean.showViewsPanel}"
    xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  <parameter id="showTaskDetailsPanel"
    value="#{testBean.showTaskDetailsPanel}"
    xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  <parameter id="wfCtxID" value="#{testBean.wfCtxID}"
    xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  <parameter id="soaHostName" value="#{testBean.soaHostName}"
    xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  <parameter id="soaPort" value="#{testBean.soaPort}"
    xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  <parameter id="refreshURL" value="#{testBean.refreshURL}"
    xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  <parameter id="localeSource" value="#{testBean.localeSource}"
    xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  <parameter id="showActionDropdown" value="#{testBean.showActionDropdown}"
    xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  <parameter id="showViewFilter" value="#{testBean.showViewFilter}"
    xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
```

```

<parameter id="showAssignmentFilter"
            value="#{testBean.showAssignmentFilter}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="showStatusFilter" value="#{testBean.showStatusFilter}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="showSearchControl" value="#{testBean.showSearchControl}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="assignmentFilter" value="#{testBean.assignmentFilter}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="viewFilter" value="#{testBean.viewFilter}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="displayColumnsList" value="#{testBean.displayColumnsList}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="sortColumn" value="#{testBean.sortColumn}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="sortOrder" value="#{testBean.sortOrder}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="taskTypesFilterList"
            value="#{testBean.taskTypesFilterList}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="attributesFilterOperator"
            value="#{testBean.attributesFilterOperator}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
<parameter id="attributesFilterList"
            value="#{testBean.attributesFilterList}"
            xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
</parameters>
    
```

[Example 33–2](#) shows the page definition code example in `consumerPageDef.xml` in which the task list is consumed as a portlet. The attribute `DefaultValue` has the value of the parameter.

Example 33–2 Task List is Consumed as a Portlet

```

<variableIterator id="variables">
  <variable Name="Worklist1_1_soaURL" Type="java.lang.Object"
    DefaultValue="${'http://<soa_host>:<soa_port>'}" />
</variableIterator>
    
```

33.4.1 Assignment Filter Constraints

The following list shows the available assignment filter constraints.

- My
- Group
- My+Group
- Reportees
- Creator
- Owner
- Reviewer
- Previous
- Admin

33.4.2 Example of File Containing All Column Constants

[Example 33-3](#) shows a file example that contains all column constants that can be passed in the `displayColumnList` parameter. The constant value must be passed. For example, for `TITLE_COLUMN = "title"`, the `"title"` must be passed, *not* the `TITLE_COLUMN`.

Example 33-3 All Column Constants That Can Be Passed in the `displayColumnList` Parameter

```
package oracle.bpel.services.workflow.repos.table;

public interface WFTaskConstants
{
    public static final String TABLE_NAME = "WFTask";
    public static final String TL_TABLE_NAME = "WFTask_TL";
    public static final String HISTORY_TABLE_NAME = "WFTaskHistory";
    public static final String HISTORY_TL_TABLE_NAME = "WFTaskHistory_TL";
    public static final String ASSIGNEE_TABLE_NAME = "WFAssignee";
    public static final String REVIEWER_TABLE_NAME = "WFReviewer";

    public static final String WFCOMMENT_TABLE_NAME = "WFComments";
    public static final String WFATTRIBUTES_TABLE_NAME = "WFMessageAttribute";
    public static final String WFATTACHMENT_TABLE_NAME = "WFAttachment";
    public static final String WFCOLLECTIONTARGET_TABLE_NAME = "WFCollectionTarget";

    //table aliases
    public static final String TABLE_ALIAS = "wfn";
    public static final String TL_TABLE_ALIAS = "wfntl";
    public static final String HISTORY_TABLE_ALIAS = "wfnh";
    public static final String HISTORY_TL_TABLE_ALIAS = "wfnhtl";
    public static final String WFCOMMENT_TABLE_ALIAS = "wfc";
    public static final String WFATTRIBUTES_TABLE_ALIAS = "wfma";
    public static final String WFATTACHMENT_TABLE_ALIAS = "wfatt";
    public static final String ASSIGNEE_TABLE_ALIAS = "wfa";
    public static final String REVIEWER_TABLE_ALIAS = "wfr";
    public static final String WFCOLLECTIONTARGET_TABLE_ALIAS = "wfct";

    //task table column
    public static final String ACCESSKEY_COLUMN = "accessKey";
    public static final String APPROVALDURATION_COLUMN = "approvalDuration";
    public static final String ACQUIRED_BY_COLUMN = "acquiredBy";
    public static final String ASSIGNEDDATE_COLUMN = "assignedDate";
    public static final String APPROVERS_COLUMN = "approvers";
    public static final String ASSIGNEES_COLUMN = "assignees";
    public static final String ASSIGNEESDISPLAYNAME_COLUMN = "assigneesDisplayName";
    public static final String REVIEWERS_COLUMN = "reviewers";
    public static final String REVIEWERSDISPLAYNAME_COLUMN = "reviewersDisplayName";
    public static final String ASSIGNEEGROUPS_COLUMN = "assigneeGroups";
    public static final String ASSIGNEEGROUPSDISPLAYNAME_COLUMN =
"assigneeGroupsDisplayName";
    public static final String ASSIGNEEUSERS_COLUMN = "assigneeUsers";
    public static final String ASSIGNEEUSERSDISPLAYNAME_COLUMN =
"assigneeUsersDisplayName";
    public static final String OUTCOME_COLUMN = "outcome";
    public static final String PARALLELOUTCOMECOUNT_COLUMN = "parallelOutcomeCount";
    public static final String PUSHBACKSEQUENCE_COLUMN = "pushbackSequence";
    public static final String CREATEDDATE_COLUMN = "createdDate";
    public static final String ELAPSEDTIME_COLUMN = "elapsedTime";
}
```

```

public static final String DIGITALSIGNATUREREQUIRED_COLUMN =
"digitalSignatureRequired";
public static final String PASSWORDREQUIREDONUPDATE_COLUMN =
"passwordRequiredOnUpdate";
public static final String SECURENOTIFICATION_COLUMN = "secureNotifications";
public static final String ENDDATE_COLUMN = "endDate";
public static final String EXPIRATIONDATE_COLUMN = "expirationDate";
public static final String EXPIRATIONDURATION_COLUMN = "expirationDuration";
public static final String IDENTITYCONTEXT_COLUMN = "identityContext";
public static final String FROMUSER_COLUMN = "fromUser";
public static final String FROMUSERDSIPLAYNAME_COLUMN = "fromUserDisplayName";
public static final String HASSUBTASK_COLUMN = "hasSubtask";
public static final String INSHORTHISTORY_COLUMN = "inShortHistory";
public static final String ISGROUP_COLUMN = "isGroup";
public static final String LANGUAGE_COLUMN = "language";
public static final String MAILSTATUS_COLUMN = "mailStatus";
public static final String MOREINFOFOROLE_COLUMN = "moreInfoRole";
public static final String NUMBEROFTIMESMODIFIED_COLUMN =
"numberOfTimesModified";
public static final String ORIGINALASSIGNEEUSER_COLUMN = "originalAssigneeUser";
public static final String REQUESTINFOUSER_COLUMN = "requestInfoUser";
public static final String STATE_COLUMN = "State";
public static final String SUBSTATE_COLUMN = "SubState";
public static final String SYSTEMSTRING1_COLUMN = "systemString1";
public static final String SYSTEMSTRING2_COLUMN = "systemString2";
public static final String SYSTEMSTRING3_COLUMN = "SystemString3";
public static final String TASKGROUPID_COLUMN = "taskGroupId";
public static final String TASKID_COLUMN = "taskId";
public static final String VERSION_COLUMN = "version";
public static final String TASKNUMBER_COLUMN = "taskNumber";
public static final String UPDATEDBY_COLUMN = "updatedBy";

public static final String UPDATEDBYDISPLAYNAME_COLUMN = "updatedByDisplayName";
public static final String UPDATEDDATE_COLUMN = "updatedAt";
public static final String UPDATEDNOTIFICATIONID_COLUMN =
"updatedNotificationId";
public static final String VERSIONREASON_COLUMN = "versionReason";
public static final String WORKFLOWPATTERN_COLUMN = "workflowPattern";
public static final String CALLBACKCONTEXT_COLUMN = "callbackContext";
public static final String CALLBACKID_COLUMN = "callbackId";
public static final String CALLBACKTYPE_COLUMN = "callbackType";
public static final String CREATOR_COLUMN = "creator";
public static final String OWNERUSER_COLUMN = "ownerUser";
public static final String OWNERGROUP_COLUMN = "ownerGroup";
public static final String OWNERROLE_COLUMN = "ownerRole";
public static final String PRIORITY_COLUMN = "priority";
public static final String DOMAINID_COLUMN = "domainId";
public static final String INSTANCEID_COLUMN = "instanceId";
public static final String PROCESSID_COLUMN = "processId";
public static final String PROCESSNAME_COLUMN = "processName";
public static final String PROCESSTYPE_COLUMN = "processType";
public static final String PROCESSVERSION_COLUMN = "processVersion";
public static final String TITLE_COLUMN = "title";
public static final String TITLERESOURCEKEY_COLUMN = "titleResourceKey";
public static final String IDENTIFICATIONKEY_COLUMN = "identificationKey";
public static final String USERCOMMENT_COLUMN = "userComment";
public static final String WORKFLOWDESCRIPTORURI_COLUMN =
"workflowDescriptorURI";
public static final String TASKDEFINITIONID_COLUMN = "taskDefinitionId";
public static final String TASKDEFINITIONNAME_COLUMN = "taskDefinitionName";

```

```

// start columns added for AS11
public static final String APPLICATIONCONTEXT_COLUMN = "applicationContext";
public static final String APPLICATIONNAME_COLUMN = "applicationName";
public static final String ASSIGNEETYPE_COLUMN = "assigneeType";
public static final String CATEGORY_COLUMN = "category";
public static final String COMPONENTNAME_COLUMN = "componentName";
public static final String COMPOSITEDN_COLUMN = "compositeDN";
public static final String COMPOSITEINSTANCEID_COLUMN = "compositeInstanceId";
public static final String COMPOSITENAME_COLUMN = "compositeName";
public static final String COMPOSITEVERSION_COLUMN = "compositeVersion";
public static final String CONVERSATIONID_COLUMN = "conversationId";
public static final String DUEDATE_COLUMN = "dueDate";
public static final String ECID_COLUMN = "ecId";
public static final String ISPUBLIC_COLUMN = "isPublic";
public static final String ISTEFTASK_COLUMN = "isTestTask";
public static final String PARENTCOMPONENTINSTANCEID_COLUMN =
"parentComponentInstanceId";
public static final String PARENTCOMPONENTINSTANCEREfid_COLUMN =
"parentComponentInstRefId";
public static final String INVOKEDCOMPONENT_COLUMN = "invokedComponent";
public static final String PARTICIPANTNAME_COLUMN = "participantName";
public static final String PERCENTAGECOMPLETE_COLUMN = "percentageComplete";
public static final String READBYUSERS_COLUMN = "readByUsers";
public static final String STARTDATE_COLUMN = "startDate";
public static final String PARENTTASKVERSION_COLUMN = "parentTaskVersion";
public static final String TASKGROUPINSTANCEID_COLUMN = "taskGroupInstanceId";
public static final String SUBTASKGROUPINSTANCEID_COLUMN =
"subTaskGroupInstanceId";
public static final String AG_ROOTID_COLUMN = "agRootId";
public static final String AG_MILESTONE_PATH_COLUMN = "agMilestonePath";
public static final String ROOTTASKID_COLUMN = "rootTaskId";
public static final String PARENTTASKID_COLUMN = "parentTaskId";
public static final String SYSTEMSTRINGACTIONS_COLUMN = "systemStringActions";
public static final String SUBCATEGORY_COLUMN = "subCategory";
public static final String CORRELATIONID_COLUMN = "correlationId";
public static final String TASKDISPLAYURL_COLUMN = "taskDisplayUrl";
public static final String STAGE_COLUMN = "stage";
public static final String ASSIGNMENTCONTEXT_COLUMN = "assignmentContext";
public static final String PREACTIONUSERSTEPS_COLUMN = "preActionUserSteps";
public static final String AGGREGATIONTASKID_COLUMN = "aggregationTaskId";
public static final String MDSLABEL_COLUMN = "mdsLabel";
public static final String ISTEMPLATETASK_COLUMN = "isTemplateTask";

/* Columns for instance locator service */
public static final String COMPONENTTYPE_COLUMN = "componentType";
public static final String ACTIVITYNAME_COLUMN = "activityName";
public static final String ACTIVITYID_COLUMN = "activityId";
public static final String PROCESSDUEDATE_COLUMN = "processDueDate";
public static final String THREAD_COLUMN = "thread";
public static final String PARENTTHREAD_COLUMN = "parentThread";
public static final String STEP_COLUMN = "step";

public static final String TASKNAMESPACE_COLUMN = "taskNamespace";
// SERVERNAME_COLUMN is pseudo column, it does not exist in the table,
// column can be used for sorting on client side by FederatedTaskQueryService in
Ordering
public static final String SERVERNAME_COLUMN = "serverName";
// end columns added for AS11
    
```

```

public static final String TEXTATTRIBUTE1_COLUMN = "textAttribute1";
public static final String TEXTATTRIBUTE2_COLUMN = "textAttribute2";
public static final String TEXTATTRIBUTE3_COLUMN = "textAttribute3";
public static final String TEXTATTRIBUTE4_COLUMN = "textAttribute4";
public static final String TEXTATTRIBUTE5_COLUMN = "textAttribute5";
public static final String TEXTATTRIBUTE6_COLUMN = "textAttribute6";
public static final String TEXTATTRIBUTE7_COLUMN = "textAttribute7";
public static final String TEXTATTRIBUTE8_COLUMN = "textAttribute8";
public static final String TEXTATTRIBUTE9_COLUMN = "textAttribute9";
public static final String TEXTATTRIBUTE10_COLUMN = "textAttribute10";
public static final String FORMATATTRIBUTE1_COLUMN = "formAttribute1";
public static final String FORMATATTRIBUTE2_COLUMN = "formAttribute2";
public static final String FORMATATTRIBUTE3_COLUMN = "formAttribute3";
public static final String FORMATATTRIBUTE4_COLUMN = "formAttribute4";
public static final String FORMATATTRIBUTE5_COLUMN = "formAttribute5";
public static final String URLATTRIBUTE1_COLUMN = "urlAttribute1";
public static final String URLATTRIBUTE2_COLUMN = "urlAttribute2";
public static final String URLATTRIBUTE3_COLUMN = "urlAttribute3";
public static final String URLATTRIBUTE4_COLUMN = "urlAttribute4";
public static final String URLATTRIBUTE5_COLUMN = "urlAttribute5";
public static final String DATEATTRIBUTE1_COLUMN = "dateAttribute1";
public static final String DATEATTRIBUTE2_COLUMN = "dateAttribute2";
public static final String DATEATTRIBUTE3_COLUMN = "dateAttribute3";
public static final String DATEATTRIBUTE4_COLUMN = "dateAttribute4";
public static final String DATEATTRIBUTE5_COLUMN = "dateAttribute5";
public static final String NUMBERATTRIBUTE1_COLUMN = "numberAttribute1";
public static final String NUMBERATTRIBUTE2_COLUMN = "numberAttribute2";
public static final String NUMBERATTRIBUTE3_COLUMN = "numberAttribute3";
public static final String NUMBERATTRIBUTE4_COLUMN = "numberAttribute4";
public static final String NUMBERATTRIBUTE5_COLUMN = "numberAttribute5";
public static final String PROTECTEDTEXTATTRIBUTE1_COLUMN =
"protectedTextAttribute1";
public static final String PROTECTEDTEXTATTRIBUTE2_COLUMN =
"protectedTextAttribute2";
public static final String PROTECTEDTEXTATTRIBUTE3_COLUMN =
"protectedTextAttribute3";
public static final String PROTECTEDTEXTATTRIBUTE4_COLUMN =
"protectedTextAttribute4";
public static final String PROTECTEDTEXTATTRIBUTE5_COLUMN =
"protectedTextAttribute5";
public static final String PROTECTEDTEXTATTRIBUTE6_COLUMN =
"protectedTextAttribute6";
public static final String PROTECTEDTEXTATTRIBUTE7_COLUMN =
"protectedTextAttribute7";
public static final String PROTECTEDTEXTATTRIBUTE8_COLUMN =
"protectedTextAttribute8";
public static final String PROTECTEDTEXTATTRIBUTE9_COLUMN =
"protectedTextAttribute9";
public static final String PROTECTEDTEXTATTRIBUTE10_COLUMN =
"protectedTextAttribute10";
public static final String PROTECTEDFORMATATTRIBUTE1_COLUMN =
"protectedFormAttribute1";
public static final String PROTECTEDFORMATATTRIBUTE2_COLUMN =
"protectedFormAttribute2";
public static final String PROTECTEDFORMATATTRIBUTE3_COLUMN =
"protectedFormAttribute3";
public static final String PROTECTEDFORMATATTRIBUTE4_COLUMN =
"protectedFormAttribute4";
public static final String PROTECTEDFORMATATTRIBUTE5_COLUMN =
"protectedFormAttribute5";

```



```

        public static final String PROTECTEDURLATTRIBUTE1_COLUMN =
        "protectedUrlAttribute1";
        public static final String PROTECTEDURLATTRIBUTE2_COLUMN =
        "protectedUrlAttribute2";
        public static final String PROTECTEDURLATTRIBUTE3_COLUMN =
        "protectedUrlAttribute3";
        public static final String PROTECTEDURLATTRIBUTE4_COLUMN
        ="protectedUrlAttribute4";
        public static final String PROTECTEDURLATTRIBUTE5_COLUMN
        ="protectedUrlAttribute5";
        public static final String PROTECTEDDATEATTRIBUTE1_COLUMN
        ="protectedDateAttribute1";
        public static final String PROTECTEDDATEATTRIBUTE2_COLUMN
        ="protectedDateAttribute2";
        public static final String PROTECTEDDATEATTRIBUTE3_COLUMN
        ="protectedDateAttribute3";
        public static final String PROTECTEDDATEATTRIBUTE4_COLUMN
        ="protectedDateAttribute4";
        public static final String PROTECTEDDATEATTRIBUTE5_COLUMN
        ="protectedDateAttribute5";
        public static final String PROTECTEDNUMBERATTRIBUTE1_COLUMN
        ="protectedNumberAttribute1";
        public static final String PROTECTEDNUMBERATTRIBUTE2_COLUMN
        ="protectedNumberAttribute2";
        public static final String PROTECTEDNUMBERATTRIBUTE3_COLUMN
        ="protectedNumberAttribute3";
        public static final String PROTECTEDNUMBERATTRIBUTE4_COLUMN
        ="protectedNumberAttribute4";
        public static final String PROTECTEDNUMBERATTRIBUTE5_COLUMN
        ="protectedNumberAttribute5";

        /*
         * Flexfield columns added for AS11
         */
        public static final String TEXTATTRIBUTE11_COLUMN = "textAttribute11";
        public static final String TEXTATTRIBUTE12_COLUMN = "textAttribute12";
        public static final String TEXTATTRIBUTE13_COLUMN = "textAttribute13";
        public static final String TEXTATTRIBUTE14_COLUMN = "textAttribute14";
        public static final String TEXTATTRIBUTE15_COLUMN = "textAttribute15";
        public static final String TEXTATTRIBUTE16_COLUMN = "textAttribute16";
        public static final String TEXTATTRIBUTE17_COLUMN = "textAttribute17";
        public static final String TEXTATTRIBUTE18_COLUMN = "textAttribute18";
        public static final String TEXTATTRIBUTE19_COLUMN = "textAttribute19";
        public static final String TEXTATTRIBUTE20_COLUMN = "textAttribute20";
        public static final String FORMATATTRIBUTE6_COLUMN = "formAttribute6";
        public static final String FORMATATTRIBUTE7_COLUMN = "formAttribute7";
        public static final String FORMATATTRIBUTE8_COLUMN = "formAttribute8";
        public static final String FORMATATTRIBUTE9_COLUMN = "formAttribute9";
        public static final String FORMATATTRIBUTE10_COLUMN = "formAttribute10";
        public static final String URLATTRIBUTE6_COLUMN = "urlAttribute6";
        public static final String URLATTRIBUTE7_COLUMN = "urlAttribute7";
        public static final String URLATTRIBUTE8_COLUMN = "urlAttribute8";
        public static final String URLATTRIBUTE9_COLUMN = "urlAttribute9";
        public static final String URLATTRIBUTE10_COLUMN = "urlAttribute10";
        public static final String DATEATTRIBUTE6_COLUMN = "dateAttribute6";
        public static final String DATEATTRIBUTE7_COLUMN = "dateAttribute7";
        public static final String DATEATTRIBUTE8_COLUMN = "dateAttribute8";
        public static final String DATEATTRIBUTE9_COLUMN = "dateAttribute9";
        public static final String DATEATTRIBUTE10_COLUMN = "dateAttribute10";
        public static final String NUMBERATTRIBUTE6_COLUMN = "numberAttribute6";
    
```

```

public static final String NUMBERATTRIBUTE7_COLUMN = "numberAttribute7";
public static final String NUMBERATTRIBUTE8_COLUMN = "numberAttribute8";
public static final String NUMBERATTRIBUTE9_COLUMN = "numberAttribute9";
public static final String NUMBERATTRIBUTE10_COLUMN = "numberAttribute10";
public static final String PROTECTEDTEXTATTRIBUTE11_COLUMN =
"protectedTextAttribute11";
public static final String PROTECTEDTEXTATTRIBUTE12_COLUMN =
"protectedTextAttribute12";
public static final String PROTECTEDTEXTATTRIBUTE13_COLUMN =
"protectedTextAttribute13";
public static final String PROTECTEDTEXTATTRIBUTE14_COLUMN =
"protectedTextAttribute14";
public static final String PROTECTEDTEXTATTRIBUTE15_COLUMN =
"protectedTextAttribute15";
public static final String PROTECTEDTEXTATTRIBUTE16_COLUMN =
"protectedTextAttribute16";
public static final String PROTECTEDTEXTATTRIBUTE17_COLUMN =
"protectedTextAttribute17";
public static final String PROTECTEDTEXTATTRIBUTE18_COLUMN =
"protectedTextAttribute18";
public static final String PROTECTEDTEXTATTRIBUTE19_COLUMN =
"protectedTextAttribute19";
public static final String PROTECTEDTEXTATTRIBUTE20_COLUMN =
"protectedTextAttribute20";
public static final String PROTECTEDFORMATATTRIBUTE6_COLUMN =
"protectedFormAttribute6";
public static final String PROTECTEDFORMATATTRIBUTE7_COLUMN =
"protectedFormAttribute7";
public static final String PROTECTEDFORMATATTRIBUTE8_COLUMN =
"protectedFormAttribute8";
public static final String PROTECTEDFORMATATTRIBUTE9_COLUMN =
"protectedFormAttribute9";
public static final String PROTECTEDFORMATATTRIBUTE10_COLUMN =
"protectedFormAttribute10";
public static final String PROTECTEDURLATTRIBUTE6_COLUMN =
"protectedUrlAttribute6";
public static final String PROTECTEDURLATTRIBUTE7_COLUMN =
"protectedUrlAttribute7";
public static final String PROTECTEDURLATTRIBUTE8_COLUMN =
"protectedUrlAttribute8";
public static final String PROTECTEDURLATTRIBUTE9_COLUMN =
"protectedUrlAttribute9";
public static final String PROTECTEDURLATTRIBUTE10_COLUMN =
"protectedUrlAttribute10";
public static final String PROTECTEDDATEATTRIBUTE6_COLUMN =
"protectedDateAttribute6";
public static final String PROTECTEDDATEATTRIBUTE7_COLUMN =
"protectedDateAttribute7";
public static final String PROTECTEDDATEATTRIBUTE8_COLUMN =
"protectedDateAttribute8";
public static final String PROTECTEDDATEATTRIBUTE9_COLUMN =
"protectedDateAttribute9";
public static final String PROTECTEDDATEATTRIBUTE10_COLUMN =
"protectedDateAttribute10";
public static final String PROTECTEDNUMBERATTRIBUTE6_COLUMN =
"protectedNumberAttribute6";
public static final String PROTECTEDNUMBERATTRIBUTE7_COLUMN =
"protectedNumberAttribute7";
public static final String PROTECTEDNUMBERATTRIBUTE8_COLUMN =
"protectedNumberAttribute8";

```

```

    public static final String PROTECTEDNUMBERATTRIBUTE9_COLUMN
    ="protectedNumberAttribute9";
    public static final String PROTECTEDNUMBERATTRIBUTE10_COLUMN
    ="protectedNumberAttribute10";

    // TL table related columns

    public static final String LOCALE_COLUMN = "locale";

    //assignee table column
    public static final String ASSIGNEE_ASSIGNEE_COLUMN = "assignee";

    public static final String WFCOMMENT_COMMENTDATE_COLUMN= "commentDate";
    public static final String WFCOMMENT_ACTION_COLUMN= "action";
    public static final String WFCOMMENT_WFCOMMENT_COLUMN= "wfcomment";
    public static final String WFCOMMENT_DISPLAYNAMELANGUAGE_COLUMN=
    "displayNameLanguage";
    public static final String WFCOMMENT_ACL_COLUMN= "acl";

    public static final String MAXVERSION_COLUMN= "maxVersion";
    public static final String WFATTRIBUTES_NAME_COLUMN= "name";
    public static final String WFATTRIBUTES_STORAGETYPE_COLUMN= "storageType";
    public static final String WFATTRIBUTES_ENCODING_COLUMN= "encoding";
    public static final String WFATTRIBUTES_STRINGVALUE_COLUMN= "stringValue";
    public static final String WFATTRIBUTES_NUMBERVALUE_COLUMN= "numberValue";
    public static final String WFATTRIBUTES_DATEVALUE_COLUMN= "dateValue";
    public static final String WFATTRIBUTES_BLOBVALUE_COLUMN= "blobValue";
    public static final String WFATTRIBUTES_ELEMENTSEQ_COLUMN= "elementSeq";

    //attachment columns
    public static final String WFATTACHMENT_ENCODING_COLUMN= "encoding";
    public static final String WFATTACHMENT_URI_COLUMN= "uri";
    public static final String WFATTACHMENT_CONTENT_COLUMN= "content";
    public static final String WFATTACHMENT_NAME_COLUMN= "name";
    public static final String WFATTACHMENT_ACL_COLUMN= "acl";

    //collection target columns
    public static final String WFCOLLECTIONTARGET_ID_COLUMN= "id";
    public static final String WFCOLLECTIONTARGET_XPATH_COLUMN= "xpath";
    public static final String WFCOLLECTIONTARGET_COLLECTIONNAME_COLUMN=
    "collectionName";
    public static final String WFCOLLECTIONTARGET_COLLECTIONNAMESPACE_COLUMN=
    "collectionNamespace";
    public static final String WFCOLLECTIONTARGET_TYPE_COLUMN= "type";
    public static final String WFCOLLECTIONTARGET_TARGETINDEX_COLUMN= "targetIndex";
    public static final String WFCOLLECTIONTARGET_KEYLIST_COLUMN= "keyList";
    public static final String WFCOLLECTIONTARGET_REFERENCEDTASKID_COLUMN=
    "referencedTaskId";
    public static final String WFCOLLECTIONTARGET_TASKAGGREGATIONID_COLUMN=
    "taskAggregationId";
    public static final String WFCOLLECTIONTARGET_ACTION_COLUMN= "action";
    public static final String WFCOLLECTIONTARGET_ACTIONPARAMS_COLUMN=
    "actionParams";

    public static final String ASSIGNEETYPE_SEPARATOR_STRING = ",";
}

```

Index

A

- abs function
 - description, B-8
- access points, 60-6, 61-9, 62-10
- access policies
 - on task content, 27-67
- action types, 27-46
- actionable emails, 31-32
 - not sent during runtime when digital signatures are enabled, 31-32
- activities
 - Annotations tab, A-4
 - assert, A-9
 - Assertions tab, 11-49, A-5
 - assign, A-6
 - bind entity, A-10
 - bypassing execution of, 10-12
 - compensate, A-11
 - compensateScope, A-12
 - Correlations tab, A-5
 - create entity, A-13
 - definition, 4-6
 - dehydrate, A-13
 - Documentation tab, A-5
 - email, A-14
 - empty, A-15
 - exit, A-16
 - flow, A-16
 - flowN, A-17
 - forEach, A-18
 - Headers tab, A-5
 - if, A-19
 - IM, A-20
 - invoke, A-20
 - Java embedding, A-21
 - overview, 4-6, A-2
 - partner link, A-22
 - phase, A-23
 - pick, A-24
 - Properties tab, A-5
 - receive, A-27
 - receive signal, A-28
 - remove entity, A-28
 - repeatUntil, A-29
 - replay, A-30
 - reply, A-30
 - rethrow, A-31
 - scope, A-32
 - sequence, A-33
 - signal, A-34
 - Skip Condition tab, 10-13, A-6
 - SMS, A-35
 - Sources tab, A-6
 - switch, A-35
 - synchronizing the execution of activities, 9-5
 - Targets tab, A-6
 - tasks common to many activities, A-4
 - terminate, A-36
 - throw, A-37
 - Timeout tab, 14-7, A-6
 - transform, A-37
 - user notification, A-38
 - validate, A-39
 - voice, A-40
 - wait, A-40
 - while, A-41
- activity sensors
 - definition, 17-2
- Adapter Configuration wizard
 - starting, 4-13
- adapters
 - binding component, 2-11, 2-17
 - configuring, 4-13
 - definition, 1-5, 4-13, 34-9, A-42
 - in Oracle JDeveloper, 4-13
 - Oracle BAM, 50-1
 - overview, 1-5, 34-9, A-42
 - service names, 4-13
 - supported, 1-5, 34-9, A-42
- add-dayTimeDuration-to-dateTime function
 - description, B-3
- adding a cross reference table column, 46-8
- adding columns to domain value maps, 44-7
- adding rows to domain value maps, 44-7
- addQuotes function
 - description, B-14
- ADF bindings
 - files for, 51-5
 - using to invoke a composite from a JSP/Java class, 34-13
- ADF bindings filter, 32-2

- ADF Model layer, introduced, 51-1
- ADF task flow for human tasks, 28-4
- ADF, using Oracle BAM, 51-4
- ADF-BC services
 - binding component, 2-11, 2-17
 - capabilities, A-42
 - definition, 34-11
- adfBindings bindings filter, 32-2
- adf-desktop-integration.jar, 32-2
- adfdiExcelDownload download filter, 32-3
- adfdiRemote servlet, 32-3
- ADFLibraryFilter filter, 32-3
- admin.server.host parameter, 3-22
- admin.server.port parameter, 3-22
- advanced formatting, message sources, 53-10
- aggregate functions in calculations, 52-5
- alerts
 - history, 57-6
- Oracle BAM
 - about, 57-1
 - actions, F-5
 - activating, 57-3
 - activity, 57-6
 - conditions, F-5
 - creating, 57-2
 - dependencies, 57-6
 - events, F-1
 - frequency constraint, F-14
 - history, 57-6
 - messages, 57-5
 - parameterized, F-7
 - templates, 57-4
 - web services, 57-8
- alidateFodConfigSettings ant script, 3-23
- Annotations tab
 - in activities, A-4
- ant scripts
 - activating all composites in a partition, 40-58
 - activating an application, 40-53
 - assigning the default version to a SOA composite application, 40-54
 - compile-deploy-all, 3-23
 - compiling a SOA composite application, 40-43
 - creating a partition in the SOA
 - Infrastructure, 40-56
 - deleting a partition, 40-57
 - deploying a SOA composite application, 40-45
 - executing a test case, 40-42
 - exporting a SOA composite application into a SAR file, 40-47
 - exporting postdeployment changes of a composite into a JAR file, 40-48
 - exporting shared data of a given pattern into a JAR file, 40-50
 - importing postdeployment changes of a composite, 40-49
 - listing all available partitions in the SOA
 - Infrastructure, 40-55
 - listing all composites in a partition, 40-55
 - listing the deployed SOA composite
 - applications, 40-54
 - managing composites, 40-41
 - packaging a SOA composite application into a composite SAR file, 40-44
 - removing a top-level shared data folder, 40-51
 - retiring all composites in a partition, 40-59
 - retiring an application, 40-53
 - seedBAMServerObjects, 3-23
 - seedDemoUsers, 3-23
 - seedFodJmsResources, 3-23
 - server-setup-seed-deploy-test, 3-23
 - starting all composites in a partition, 40-57
 - starting an application, 40-51, 40-52, 40-53, 40-54
 - stopping all composites in a partition, 40-58
 - stopping an application, 40-52
 - undeploying a SOA composite application, 40-46
 - validateFodConfigSettings, 3-23
- appendToList function
 - description, B-16
- Application Navigator
 - location of in Oracle JDeveloper, 4-4
- application roles
 - definition, 26-5
- application template, 32-2
- AQ adapter
 - capabilities, A-42
 - definition, 34-9
- arrays
 - in transformations, 37-28
 - manipulating, 6-45
 - maxOccurs attribute, 6-45
 - SOAP-encoded arrays, 6-46
 - statically indexing into, 6-45
- assert activity
 - capabilities, A-9
 - only supported in BPEL 1.1 projects, A-9
- assertion conditions
 - creating, 11-49
 - disabling, 11-52
 - expressions not evaluating to an XML schema
 - boolean type throw a fault, 11-49
 - log events in the instance audit trail, 11-49
 - multiple, 11-47
 - throwing faults, 11-45
 - use of built-in and custom XPath functions and \$variable references, 11-48
- assertion tests
 - overview, 41-2
- assertions
 - creating value asserts, 41-18
 - in composite test suites, 41-4
- Assertions tab
 - creating assertion conditions, 11-49
 - in activities, A-5
 - only available in BPEL 1.1 projects, A-5
- assign activity
 - adding to an asynchronous service, 8-5
 - assigning a literal or XML fragment to a target node, A-7
 - bpelx extensions in BPEL 1.1, 6-22

- bpelx extensions in BPEL 2.0, 6-23
- capabilities, A-6
- changing copy rules to bpelx extension types, A-7
- copying data, 6-14, A-6
- creating a bpelx:rename extension rule on a target node, A-7
- creating an XPath expression on a target node, A-7
- description, 6-3
- for data manipulation, 6-2, A-6
- formatting the email message body as HTML, 16-8
- in asynchronous services, 8-5
- recasting a target node, A-7
- renaming a target node, A-7
- selecting an extension type in BPEL 1.1, A-8
- selecting an extension type in BPEL 2.0, A-8
- selecting the ignoreMissingFromData attribute, A-9
- selecting the insertMissingToData attribute, A-9
- selecting the keepSrcElementName attribute, A-9
- using multiple bpelx:append settings, A-9
- using the Copy Rules tab, 6-22, A-6
- assign extension attributes
 - ignoreMissingFromData, 6-34
 - insertMissingToData, 6-34
 - keepSrcElementName, 6-34
 - using, 6-34
- assignment service
 - configuration, 31-36
 - deploying a custom assignment service, 31-43
 - dynamic assignment functions, 31-37, 31-38, 31-39
 - dynamically assigning task participants, 31-39
 - example of implementation, 31-41
 - implementing, 31-40
- asynchronous interaction with a notification timer
 - BPEL process as the client, 5-6
 - BPEL process as the service, 5-6
 - definition, 5-5
- asynchronous interaction with a timeout
 - BPEL process as the client, 5-5
 - BPEL process as the service, 5-5
 - definition, 5-4
- asynchronous interactions
 - BPEL process as the client, 5-4
 - BPEL process as the service, 5-4
 - definition, 5-3
 - returning faults, 11-28
- asynchronous processes
 - dehydration store, 8-9
- asynchronous services
 - assign activities, 8-5
 - calling, 8-2
 - correlation IDs, 8-8
 - invoke activities, 8-3, 8-8
 - parallel flows, 9-1
 - partner links, 8-2, 8-6, 8-7
 - receive activities, 8-4, 8-8

- WS-Addressing, 8-8
- attachments
 - adding MTOM attachments to web services, 42-9
 - attaching a URL file, 29-35
 - in end-to-end streaming, 42-2
 - in SOAP, 42-3
 - MIME, 42-4
 - optimization enabled, 42-6
 - options for file and FTP adapters, 42-8
 - Oracle B2B, 42-9
 - performance overhead and pass through attachments, 42-6
 - properties for streaming attachments, 42-6
 - reading and encoding SOAP attachment content, 42-7
 - sending streaming attachments, 42-7
 - sending with the notification wizard, 16-7
 - sharing attachments using synchronous flows, 42-7
 - streaming, 42-3
 - task attachments with email notifications, 31-34
 - transforming attachments with the ora:doStreamingTranslate XPath function, 42-8
 - using WordML style sheets, 27-54
 - writing attachments using an outbound file adapter, 42-8
- attribute labels
 - internationalization, 31-18
- attributes
 - manipulating, 6-21
- audit level
 - setting, 42-12
- authenticate function
 - description, B-14
- auto mapping
 - in transformations, 37-32
 - with confirmation in transformations, 37-33
- automated testing
 - of SOA composite applications, 2-29, 41-1

B

- B2B*See* Oracle B2B
- B2BX12OrderGateway project, 3-7
- bam.server.host parameter, 3-21
- bam.server.password parameter, 3-21
- bam.server.port parameter, 3-21
- bam.username parameter, 3-21
- batching
 - message batching limitations with Oracle Business Activity Monitoring, 50-32
- batchProcessActive function
 - description, B-42
- batchProcessCompleted function
 - description, B-43
- best practices
 - creating and wiring BPEL and mediator service components in the SOA Composite Editor, 4-9

- for handling large documents, 42-1
 - for handling large metadata, 42-18
 - for handling large numbers of instances, 42-20
 - tuning recommendations, 42-12
- bin project, 3-7
- bind activity
 - only supported in BPEL 1.1 projects, A-10
- bind entity activity
 - capabilities, A-10
- binding components
 - adapters, 2-11, 2-17
 - adding references, 2-16
 - adding services, 2-10
 - ADF-BC services, 2-11, 2-17, 34-11
 - definition, 1-5, 1-7
 - deleting references, 2-18
 - deleting services, 2-16
 - direct binding services, 2-11, 2-17, 34-12
 - editing, 2-16
 - EJB services, 2-11, 2-17, 34-11
 - HTTP binding, 2-11, 2-17, 34-5
 - integrating into a SOA composite
 - application, 34-12
 - introduction, 34-1
 - JCA adapters, 34-9
 - Oracle B2B, 2-11, 2-17, 34-11
 - Oracle BAM, 34-11
 - supported, 1-5, 2-11
 - web services, 2-11, 2-17, 34-2
 - WS-Atomic transactions, 34-2
- bindingFault
 - definition, 11-6
- boolean values
 - assigning, 6-19
- bottom-up design approach, 1-8
- bpel
 - BPEL 2.0 namespace prefix, 6-4, B-40
- BPEL 1.1
 - activities overview, A-2
- BPEL 2.0
 - \$variable syntax, 6-18
 - activities overview, A-2
 - assign activity, 6-3
 - assigning a date or time, 6-20
 - assigning boolean values, 6-19
 - bpelx:append extension, 6-24
 - bpelx:copyList extension, 6-33
 - bpelx:insertAfter extension, 6-27
 - bpelx:insertBefore extension, 6-25
 - bpelx:remove extension, 6-29
 - bpelx:rename extension, 6-31
 - cannot create a single BPEL project that supports
 - versions 1.1 and 2.0, 4-2
 - compensateScope activity, 11-41, A-12
 - conditional branching, 10-5
 - creating a BPEL project, 4-2
 - custid attribute, 6-21
 - declaring extension namespaces, 6-55
 - determining the BPEL project version
 - number, 4-5
 - element-based variables, 6-17
 - exit activity, 11-43, A-16
 - expression copy, 6-18
 - forEach activity, 9-16, A-18
 - fromParts element, 6-37
 - if activity, 10-5, A-19
 - importing process definitions, 6-44
 - initializing variables inline, 6-15, 8-5
 - limitations
 - on inbound message activity support, 8-9
 - mapping WSDL message parts, 6-37
 - message type-based variable definition, 6-16
 - namespace prefix, 6-4
 - order of precedence for fault handling, 11-4
 - repeatUntil activity, 10-10, A-29
 - rethrow activity, 11-26, A-31
 - setting correlations for an IMA using a fromParts
 - element with multiple parts, 8-26
 - simultaneous onMessage branches, 14-6
 - SOAP-encoded arrays, 6-47
 - standard faults, 11-3
 - supported activities, A-2
 - toParts element, 6-37
 - using element variables in message exchange
 - activities, 6-36
 - waiting for message arrival with an onEvent
 - branch, 14-13
- BPEL design environment
 - overview, 4-1
- BPEL extension functions
 - in BPEL 1.1, B-40
 - in BPEL 2.0, B-40
- BPEL monitors
 - definition, 4-14
- BPEL processes
 - common interaction patterns, 5-1, 23-1
 - creating, 3-13, 3-14, 4-1
 - definition, 1-3
 - naming conventions, 4-2
 - service component, 2-7
 - supported versions, 1-3
 - transaction semantics, 12-1
- BPEL projects
 - determining the BPEL project version
 - number, 4-5
 - naming conventions, 4-2
- BPEL sensor
 - Oracle BAM, 50-29
- BPEL XPath extension functions, 6-5, B-14
 - examples, 6-4
- bpelx extensions
 - bpel:rename, A-7
 - bpelx:append, 6-23, 6-24, 6-46, 6-49, 16-9, A-7
 - bpelx:append extension, 6-23
 - bpelx:conversationId, 8-10
 - bpelx:copyList, 6-32, 6-33, A-7
 - bpelx:dehydrate name, A-13
 - bpelx:detailLabel, 15-5
 - bpelx:exec, 13-2, 13-8
 - bpelx:flowN, 9-15

- bpelx:for, 14-7
- bpelx:fromProperties, H-5
- bpelx:headerVariable, 6-53
- bpelx:ignoreMissingFromData, 6-34
- bpelx:inputProperty, H-4
- bpelx:insertAfter, 6-26, 6-27, A-7
- bpelx:insertBefore, 6-24, 6-25, A-7
- bpelx:insertMissingToData, 6-34
- bpelx:invokeAsDetail, 15-4
- bpelx:outputProperty, H-4
- bpelx:postAssert, 11-45
- bpelx:preAssert, 11-45
- bpelx:receiveSignal, 15-5
- bpelx:remove, 6-13, 6-28, 6-29, A-7
- bpelx:rename, 6-29, 6-31, A-7
- bpelx:rollback, 12-3
- bpelx:sdoCapable, 6-11
- bpelx:signal, 15-4
- bpelx:skipCondition, 10-12
- bpelx:target, 6-13, 6-30
- bpelx:timeout, 14-9
- bpelx:until, 14-8
- bpelx:validate, 6-35
 - in assign activities, A-9
 - in assign activities in BPEL 1.1, 6-22
 - in assign activities in BPEL 2.0, 6-23
 - XML data manipulation, 6-22
- bpelx:append extension
 - appending data to a node list, B-16
 - appending new items in a sequence, 6-49
 - changing a copy rule to, A-7
 - description, 6-23
 - email activity message content, 16-9
 - in SOAP-encoded arrays, 6-46
 - not supported for SDO-based variables, 6-13
 - using, 6-23, 6-24
- bpelx:assert extension
 - expressions not evaluating to an XML schema
 - boolean type throw a fault, 11-49
 - multiple assertions, 11-47
 - throwing faults based on a condition, 11-45
 - use of built-in and custom XPath functions and \$variable references, 11-48
 - use of faultName and message attributes, 11-47
- bpelx:conversationId extension, 8-10
- bpelx:copyList extension
 - changing a copy rule to, A-7
 - copying a node list or a node, B-16
 - description, 6-31
 - using, 6-32, 6-33
- bpelx:dehydrate name extension
 - description, A-13
- bpelx:detailLabel extension, 15-5
- bpelx:exec extension, 13-2, 13-8
 - built-in methods, 13-4
 - embedding SDOs, 13-8
- bpelx:flowN extension, 9-15
- bpelx:for extension, 14-7
- bpelx:fromProperties extension, H-5
- bpelx:ignoreMissingFromData extension
 - using, 6-34
- bpelx:headerVariable extension, 6-53
 - description, 6-53
- bpelx:inputProperty extension, H-4
- bpelx:insertAfter extension
 - changing a copy rule to, A-7
 - description, 6-26
 - using, 6-26, 6-27
- bpelx:insertBefore extension
 - changing a copy rule to, A-7
 - description, 6-24
 - using, 6-24, 6-25
- bpelx:insertMissingToData extension
 - using, 6-34
- bpelx:invokeAsDetail extension, 15-4
- bpelx:outputProperty extension, H-4
- bpelx:postAssert extension, 11-45
- bpelx:preAssert extension, 11-45
- bpelx:receiveSignal extension, 15-5
- bpelx:remove extension
 - creating, A-7
 - description, 6-27
 - using, 6-28, 6-29
- bpelx:rename extension
 - adding, A-7
 - description, 6-29
 - using, 6-29, 6-31
- bpelx:rollback extension, 12-3
- bpelx:sdoCapable extension
 - declaring SDO-based variables, 6-11
- bpelx:signal extension, 15-4
- bpelx:skipCondition extension
 - bypassing activity execution, 10-12
 - not supported in BPEL 2.0, 10-12
 - specifying to bypass activities, 10-12
- bpelx:target extension, 6-13, 6-30
- bpelx:timeout extension, 14-9
 - fault thrown during an activity timeout, 14-9
- bpelx:until extension, 14-8
- bpelx:validate extension
 - description, 6-35
 - using in BPEL 1.1, 6-35
- bpws
 - BPEL 1.1 namespace prefix, 6-4, B-40
- building expression with domain value map
 - functions, 44-11
- build.properties file
 - WebLogic Fusion Order Demo
 - build.properties file, 3-21
- business events
 - creating, 38-3
 - creating an Oracle Mediator for an event
 - subscription, 18-20
 - definition, 38-1
 - differences with direct service invocations, 38-2
 - Event Delivery Network, 38-3
 - local and remote boundaries, 38-3
 - publishing, 38-9
 - specifying callback classes, 27-75
 - subscribing to, 38-6, 38-13

- business faults
 - definition, 11-5
- business process instance
 - stopping, 11-42
- business rules
 - action types, 27-46
 - declarative components and task flows, 25-1
 - design environment overview, 24-5
 - fact types, 27-45
 - linked dictionary support, 27-48
 - OrderBookingComposite, used in, 3-10
 - routing policies, 27-41
 - service component, 2-7, 24-13
 - specifying advanced routing rules, 27-44
 - use case for data validation and constraint checks, 24-2
 - use case for dynamic processing, 24-2
 - use case for externalizing decision points in the process, 24-2
 - use case for human workflow, 24-2
 - use cases, 24-2
 - using the business rules dictionary editor
 - declarative component, 25-24
 - using the declarative component, 25-2
- Business Rules Designer
 - introduction, 24-2
 - layout, 24-2

C

- calculated fields, 52-5
- calculations
 - aggregate functions, 52-5
 - datetime functions, 52-5
 - expressions, 52-5
 - string functions, 52-5
- callback classes
 - specifying business events, 27-75
 - specifying on task status, 27-74
- callbacks
 - class loading, 31-44
 - task routing and customization in BPEL
 - callbacks, 27-77
 - using with spring, 49-4
 - viewing, 27-87
- case sensitivity
 - human workflow, 31-47
 - in group names, 27-12
- catch branch
 - creating, 11-34, A-33
 - definition, 11-33
 - fault handling, 11-29
- catchAll branch
 - definition, 11-33
- channels
 - email, 16-4
 - IM, 16-9
 - SMS, 16-10
 - voice mail, 16-12
- character set encoding
 - changing, 27-65
- chunking
 - with the file and FTP adapters, 42-11
- class paths
 - for clients using remote Enterprise
 - JavaBeans, 30-6
 - for clients using SOAP, 30-6
- clearing data objects, 52-18
- clearTaskAssignees function
 - description, B-58
- compare function
 - description, B-8
- compare-ignore-case function
 - description, B-9
- compensate activity
 - capabilities, A-11
 - definition, 11-39
 - fault handling, 11-39
- compensateScope activity
 - capabilities, A-12
 - only supported in BPEL 2.0 projects, A-12
 - using, 11-41
- compilation
 - increasing memory to recover from errors, 40-66
- compile-deploy-all ant script, 3-23
- completionPersistPolicy property
 - description, C-2
- complex structures
 - processing large XML documents, 42-11
- complex type
 - variables, 6-15
- Component Palette
 - introduction, 2-5
 - location of in Oracle JDeveloper, 4-4
- componentType file
 - definition, 2-4
- composite sensors
 - adding, 47-2
 - adding a property, 47-7
 - adding a variable, 47-6
 - adding an expression, 47-6
 - definition, 47-1
 - monitoring during runtime, 47-8
 - restrictions on use, 47-1
- composite test
 - assertions overview, 41-2
 - creating test suites, 41-5
 - creating value asserts, 41-18
 - definition, 41-1
 - deploying test suites, 41-23
 - emulating inbound messages, 41-9
 - emulations overview, 41-2
 - naming limitations on test suites and test cases, 41-5
 - test case overview, 41-1
 - test suite assertions, 41-4
 - test suite components, 41-3
 - test suite emulations, 41-3
 - test suites overview, 41-1
 - test suites process initiation, 41-3

- XML assert, 41-2
- composite.xml file
 - definition, 2-4, 2-6
 - deployment descriptors, C-1, C-3
 - opening through a SOA-MDS connection, 2-6
 - registering sensors and sensor actions, 17-14
 - syntax, 2-22
- concat function
 - description, 6-18
- conditional branching logic
 - definition, 10-1
 - use of XPath expressions, 10-1
 - using switch activities, 10-2
 - using while activities, 10-8
- conditional processing
 - with xsl choose, 37-28
 - with xsl if, 37-26
- configuration plans
 - creating, 40-11
 - creating with the WLST utility, 40-14
 - definition, 40-7
 - use cases, 40-10
- configuration properties
 - deployment descriptors, C-1
- connections
 - creating a SOA-MDS connection, 40-35
 - creating an application server connection, 40-15
 - opening the composite.xml through a SOA-MDS connection, 2-6
 - Oracle BAM Server, 50-25, 51-2
- constant values
 - in transformations, 37-18
- conversation ID
 - adding, 8-10
- Copy Rules tab
 - using in an assign activity, A-6
- copying security filters, 52-15
- copyList function
 - description, B-16
- core XPath functions
 - examples, 6-4
- correlation ID
 - WS-Addressing, 8-8
- correlation sets
 - associating with receive activities, 8-22
 - creating, 8-21
 - creating property aliases, 8-23
- correlations
 - adding on an OnMessage branch of a pick activity, A-26
 - setting for an IMA using a fromParts element with multiple parts, 8-26
 - using in an asynchronous service, 8-15
- Correlations tab
 - in activities, A-5
 - using, 8-22, 8-23
- countNodes function
 - description, B-17
- create domain value maps, 44-4
- create entity activity
 - capabilities, A-13
 - only supported in BPEL 1.1 projects, A-13
- create instance
 - definition, 8-8
 - in receive activities, 8-8
- create-delimited-string function
 - description, B-9
- createInstance attribute, 8-8
- create-nodeset-from-delimited-string function
 - description, B-52
- createWordMLDocument function
 - description, B-58
- creating cross reference tables, 46-4
- creating folders for data objects, 52-10
- creating mediator component
 - mediator files, 18-4
- CreditCardAuthorization project, 3-7
- cross reference table look up, 46-15
 - xref
 - lookupXRef function, 46-15
- cross reference tables, 46-1
 - adding a column, 46-8
 - creating, 46-4
 - deleting values, 46-19
 - looking up, 46-15
 - modifying, 46-4
 - populating columns, 46-9
 - xref
 - lookupXRef function, 46-15
 - markForDelete function, 46-19
 - populateXRefRow1M function, 46-12
- cross references
 - creating, 46-4
 - introduction, 46-1
 - modifying, 46-4
 - overview, 46-1
- current-date function
 - description, B-3
- current-dateTime function
 - description, B-4
- current-time function
 - description, B-4
- custom classes
 - adding to a SOA composite application, 13-6
- custom escalation function
 - using, 31-43
- custom sensors
 - publish type, 17-2
- Custom Task Form Wizard
 - creating a task display form, 28-12
- customization
 - adding XSD or WSDL files, 43-4
 - compiling and deploying a customized application, 43-7
 - creating a customized SOA composite application, 43-2
 - editing artifacts in a customized composite, 43-5
 - linked business rule dictionary support, 27-48
 - of SOA composite applications, 43-1
 - resolving a sequence conflict, 43-6

- resolving validation errors in Oracle JDeveloper, 43-5
- searching for customized activities, 43-4
- the customer SOA composite application, 43-9
- the vertical SOA composite application, 43-7
- upgrading the SOA composite application, 43-11

D

- data control, Oracle BAM
 - about, 51-1
 - aggregates, 51-16
 - calculated fields, 51-9
 - creating, 51-4
 - field selection, sorting, 51-11
 - filters, 51-11
 - flat query, 51-6
 - group query, 51-6
 - groups, 51-15
 - parameters, 51-7
 - query type, 51-5
 - time groups, 51-15
- data controls
 - creating, 51-4
 - displayed on the Data Controls panel, 51-5
- Data Controls panel
 - icons defined, 51-5
 - using to create a user interface, 51-5
- data manipulation
 - accessing fields with complex type variables, 6-15
 - assigning boolean values, 6-19
 - assigning date or time, 6-20
 - assigning literal strings, 6-18
 - assigning numeric values, 6-17
 - concatenating strings, 6-18
 - converting from a string to a structured XML object type, 6-51
 - copying data between variables, 6-14
 - dynamically indexing into a data sequence, 6-48
 - generating array-equivalent functionality with the `genEmptyElem` function, 6-50
 - initializing variables, 6-13
 - manipulating arrays, 6-45
 - manipulating attributes, 6-21
 - mathematical calculations with XPath functions, 6-17
 - statically indexing into a data sequence, 6-45
 - with assign activities, 6-2, 6-14
 - with XQuery and XSLT, 6-5
- data objects
 - about, 52-1
 - adding dimensions, 52-15
 - calculated column, 52-5
 - clearing contents, 52-18
 - contents, 52-9
 - creating folders, 52-10
 - datetime column, 52-6
 - defining, 52-2
 - deleting, 52-18
 - dimensions, 52-15
 - general information, 52-8
 - indexes, 52-17
 - layout, 52-9
 - lookup column, 52-4
 - moving, 52-17
 - Oracle Data Integrator, 52-6
 - organizing, 52-10
 - permissions, 52-6
 - folders, 52-11
 - renaming, 52-17
 - security filters, 52-13
 - system, 52-6
 - viewing, 52-8
- data sequences
 - determining the size, 6-47
 - dynamically indexing into, 6-48
- database
 - sensor publish type, 17-2
- database adapter
 - capabilities, A-43
 - definition, 34-9
- database views
 - human workflow, 31-60
- DataObjectDefinition web service, 56-3
- DataObjectOperations web service, 56-2
- date time stamp field, 52-6
- dates
 - assigning, 6-20
- datetime functions in calculations, 52-5
- day-from-dateTime function
 - description, B-4
- db.adminUser parameter, 3-15
- db.demoUser.tablespace parameter, 3-16
- debatching
 - debatching with the file and FTP adapters, 42-10
- declarative components
 - definition, 25-1
 - using, 25-2
 - using the business rules dictionary editor
 - declarative component, 25-24
- defining a fault handler, 11-24
- dehydrate activity
 - capabilities, A-13
- dehydration store, 8-9
 - definition, 8-9
- deleting cross reference table value, 46-19
 - xref
 - markForDelete function, 46-19
- deleting data objects, 52-18
- deleting folders, 52-12
- deployment
 - anatomy of a composite, 40-3
 - common configuration plan issues to check, 40-64
 - common deployment issues to check, 40-62
 - creating an application server connection, 40-15
 - customizing your application for the target environment, 40-7
 - in a partition, 40-24
 - invoking other deployed composites, 2-25

- managing deployed composites, 2-26
 - of a single composite, 40-14
 - of a task flow, 40-21
 - of an existing archive, 40-39
 - of multiple composites, 40-27
 - of shared metadata across composites, 40-29
 - of SOA composite applications, 2-25, 2-28
 - packaging of artifact files, 40-2
 - postdeployment configuration, 40-61
 - preparing the target environment, 40-3
 - prerequisites, 40-2
 - releasing locks to resolve ADF task form EAR file
 - deployment errors, 40-65
 - to a cluster, 40-61
 - to a managed Oracle WebLogic Server, 40-64
 - to a SAR, 40-18
 - to a two-way, SSL-enabled Oracle WebLogic Server
 - is not supported, 40-65
 - to an application server, 40-18
 - troubleshooting, 40-62
 - with an unreachable proxy server, 40-65
 - with the ant scripts, 40-41
 - with the WLST utility, 40-41
 - deployment descriptor file
 - See web.xml file*
 - deployment descriptors
 - composite.xml file, C-1, C-3
 - configuration properties, C-1
 - defining a configuration property, C-1
 - deprecated, C-3
 - overview of properties, K-1
 - Designer window
 - location of in Oracle JDeveloper, 4-4
 - dictionaries
 - in transformations, 37-36
 - limitation on generating dictionaries that use
 - functions, 37-37
 - linked dictionary support, 27-48
 - digital signatures, 31-19
 - acting on tasks that require a signature, 29-36
 - actionable emails not sent during runtime, 31-32
 - specifying, 27-71
 - dimensions
 - adding to data objects, 52-15
 - data object, 52-15
 - time, 52-16
 - direct binding invocation API, 36-3
 - direct binding service
 - asynchronous direct binding invocation, 36-4
 - binding component, 2-11, 2-17
 - capabilities, A-43
 - definition, 1-5, 34-12
 - direct binding invocation API, 36-3
 - invoking Oracle Service Bus (OSB), 34-12, 36-6, A-43
 - not recommended for processing large
 - documents, 42-3
 - overview, 36-3
 - samples using the invocation API, 36-12
 - SOA direct address syntax, 36-5
 - SOA transaction propagation, 36-5
 - synchronous direct binding invocation, 36-4
 - disableAsserts property
 - description, C-2, K-1
 - doc function
 - description, B-17
 - Documentation tab
 - in activities, A-5
 - only available in BPEL 2.0 projects, A-5
 - domain value maps
 - add columns, 44-7
 - add rows, 44-7
 - committing changes at runtime with the SOA
 - Composer, 45-5
 - creation, 44-4
 - dvm
 - lookupValue function, 44-8
 - lookupValue1M function, 44-9
 - editing, 44-7
 - editing at runtime with the SOA Composer, 45-1, 45-4
 - features, 44-2
 - one-to-many mapping, 44-4
 - qualifier order, 44-3
 - qualifiers, 44-2
 - one-to-many mapping, 44-4
 - qualifier order, 44-3
 - qualifiers, 44-2
 - saving at runtime with the SOA Composer, 45-5
 - using, 44-8
 - using in a transformation, 44-9
 - using lookupValue functions, 44-11
 - viewing at runtime with the SOA Composer, 45-3
 - domain value maps functions
 - dvm
 - lookupValue, 44-8
 - lookupValue1M, 44-9
 - domain value maps qualifiers, 44-2
 - download filter, 32-3
 - dvm
 - lookupValue function, 44-8
 - lookupValue1M function, 44-9
 - dynamic assignment functions
 - configuring, 31-38
 - configuring display names, 31-39
 - definition, 31-37
 - implementing, 31-37
 - dynamic partner links
 - using, 8-11
 - dynamic routing decision table
 - using with two-layer business process
 - management, 48-6
- ## E
-
- EclipseLink O/X Mapper (OXM)
 - See OXM*
 - edit domain value maps
 - add columns, 44-7
 - add rows, 44-7

- EDN
 - See* Event Delivery Network
- EJB services
 - binding component, 2-11, 2-17
- elements
 - ignoring in XSLT documents, 37-41
- email
 - dynamically setting addresses, 16-12
 - making emails actionable, 31-32
 - notifications support, 16-2, 16-4
- email activity
 - capabilities, A-14
 - notification support, 16-5
- email attachments
 - notifications support, 16-7
- email messages
 - HTML content for message body, 16-8
 - using dynamic HTML for message content
 - requires a CDATA function, 16-9
- empty activity
 - capabilities, A-15
 - definition, 11-36
 - fault handling, 11-36
- emulation tests
 - overview, 41-2
- emulations
 - emulating inbound messages, 41-9
 - in BPEL test suites, 41-3
- enable.bam.sensors parameter, 3-20
- ending
 - tasks, 27-55
- endpoint locations
 - multiple, 8-9
- endpointURI
 - property, K-3
- ends-with function
 - description, B-9
- Enterprise JavaBeans
 - capabilities in SOA composite applications, A-43
 - creating an Enterprise JavaBeans service, 1-5, 35-8, A-43
 - integrating Java interfaces with SOA composite applications, 35-2
 - interacting with SOA composite applications, 35-1
 - support in workflow services, 31-2
 - supported versions, 35-1
- Enterprise JavaBeans (EJB) service
 - creating an Enterprise JavaBeans service, 34-11
- enterprise message sources
 - about, 53-1
 - creating, 53-2
 - datetime specification, 53-7
 - defining, 53-2, 55-2
 - XML formatting, 53-10
- entity variable
 - binding key, 6-9
 - creating, 6-7
 - definition, 6-6
 - supported in BPEL 1.1 projects only, 6-5
 - using, 6-5
- error assignee
 - configuring, 27-51
 - definition, 26-7
- errors
 - invalid settings, A-55
- escalating
 - tasks, 27-55
- escalation policy
 - escalate after, 27-58
 - overview, 27-56
 - specifying, 27-59
- evaluation time
 - definition, 17-5
- Event Delivery Network
 - business events published in, 38-3
 - EDN-DB, 38-3
 - EDN-JMS, 38-3
 - implementations, 38-3
- evidence store service, 31-19
 - definition, 31-19
 - Enterprise JavaBeans, SOAP, and Java support, 31-2
 - WSDL file location, 31-3
- Excel workbook
 - MIME mapping, 32-3
- exceptions, 11-5
- exit activity
 - capabilities, A-16
 - immediately ending a business process
 - instance, 11-43
 - replaces the terminate activity in BPEL 2.0, A-2
- EXM
 - support in SOA composite applications, 49-29
- expiration policy
 - expire after, 27-57
 - never expire, 27-57
 - overview, 27-56
 - renew after, 27-58
- export file sample
 - ICommand, G-18
- expression builder dialog
 - using domain value map functions, 44-11
- expression constants
 - variable initialization, 6-13
- expressions in calculations, 52-5
- extended mapping (EXM)
 - See* EXM
- extension namespaces
 - declaring in BPEL 2.0, 6-55
- external data source
 - about, 55-1
 - creating, 55-2
 - Oracle Data Integrator, 55-2
- external routing
 - routing policy, 27-49
- ExternalLegacyPartnerSupplier project, 3-7

F

facets

- in the task display form, 28-13

fact types, 27-45

fault bindings, 21-8

fault handling, 11-24

- creating, 11-1, 11-24

- definition, 11-1

- fault policy, 11-6

- importing RuntimeFault.wsdl, 11-24

- modifying the WSDL files, 11-24

- order of precedence in BPEL 2.0, 11-4

- returning external faults, 11-28

- specifying an assertion condition, 11-45

- throwing internal faults, 11-25

- using catch branches, 11-29

- using compensate activities, 11-39

- using empty activities, 11-36

- using scope activities, 11-29

- using terminate activities, 11-43

- using the getFaultAsString function, 11-25

- using throw activities, 11-25

fault management framework

- associating a fault policy with a fault policy binding, 11-12

- definition, 11-6

- designing, 11-7

- executing a fault policy, 11-17

- using a Java action fault policy, 11-17

fault policy, 21-1

- actions, 21-4

- associating with a fault policy binding, 11-12

- component level, 21-8

- composite level, 21-8

- conditions, 21-2

- definition, 11-6

- designing, 11-7

- executing, 11-17

- sample file, 11-11

- using a Java action fault policy, 11-17

fault policy bindings

- sample file, 11-16

fault sensors

- definition, 17-2

fault-bindings.xml, 21-15

- fault policy bindings file, 11-7

fault-policies.xml, 21-11

- fault policy file, 11-7

faults

- categories of faults in BPEL, 11-5

- Qname fault name, 11-5

- returning external faults, 11-28

- standard faults, 11-3

- throwing internal faults, 11-25

- throwing with assertion conditions, 11-45

fields

- calculated, 52-5

- lookup, 52-4

- timestamp, 52-6

file adapter

- capabilities, A-43

- chunking, 42-11

- debatching, 42-10

- definition, 34-9

- streaming, 42-11

filters

- adfBindings, 32-2

- adfdiExcelDownload, 32-3

- ADFLibraryFilter, 32-3

- bindings filter, 32-2

- copying, 52-15

- Oracle BAM security, 52-13

fire and forget

- one-way message, 5-1

flex fields

- See* mapped attributes

flow activity

- capabilities, A-16

- creating a parallel flow, 9-2

- link synchronization syntax differences between BPEL 1.1 and 2.0, 9-5

- synchronizing the execution of activities, 9-5

flowN activity

- capabilities, A-17

- customizing the number of flow activities, 9-11

- definition, 9-11

- replaced by the forEach activity in BPEL 2.0, A-3

fod.application.issuaenabled property, 3-17

folder permissions, 52-11

folders

- deleting, 52-12

- renaming, 52-12

forEach activity

- capabilities, A-18

- processing multiple sets of activities, 9-16

- replaces the flowN activity in BPEL 2.0, A-3

- successfulBranchesOnly attribute is not

- supported, 9-18

foreign.mds.type parameter, 3-22

format function

- description, B-43

formatDate function

- description, B-21

format-dateTime function

- description, B-5

format-string function

- description, B-10

FTP adapter

- capabilities, A-43

- chunking, 42-11

- debatching, 42-10

- definition, 34-10

- streaming, 42-11

functions

- abs, B-8

- add-dayTimeDuration-to-dateTime, B-3

- addQuotes, B-14

- advanced, B-52

- appendToList, B-16

- authenticate, B-14

- batchProcessActive, B-42
- batchProcessCompleted, B-43
- BPEL XPath extension, B-14
- chaining in transformations, 37-20
- clearTaskAssignees, B-58
- compare, B-8
- compare-ignore-case, B-9
- concat, 6-18
- copyList, B-16
- countNodes, B-17
- create-delimited-string, B-9
- create-nodeset-from-delimited-string, B-52
- createWordMLDocument, B-58
- creating user-defined XPath extension
 - functions, B-74
- current-date, B-3
- current-dateTime, B-4
- current-time, B-4
- day-from-dateTime, B-4
- descriptions, 37-19
- doc, B-17
- dynamically setting email addresses and telephone
 - numbers, 16-12
- editing in transformations, 37-20
- editing XPath expressions in
 - transformations, 37-24
- ends-with, B-9
- examples, 6-4
- format, B-43
- formatDate, B-21
- format-dateTime, B-5
- format-string, B-10
- functions prefixed with xp20 or orcl, 37-19
- genEmptyElem, 6-50, B-43
- generateGUID, B-21
- generate-guid, B-53
- getChildElement, B-44
- getContentAsString, B-23
- get-content-as-string, B-10
- getConversationId, B-23
- getCreator, B-24
- getCurrentDate, 6-20, B-24
- getCurrentDateTime, 6-20, B-24
- getCurrentTime, 6-20, B-24
- getDefaultRealmName, B-63
- getDomainId, B-25
- getElement, B-25
- getFaultAsString, 11-25
- getGroupIdsFromGroupAlias, B-26
- getGroupProperty, B-63
- getInstanceId, B-26
- getLinkStatus, B-40
- get-localized-string, B-11
- getManager, B-63
- getMessage, B-44
- getNodes, B-27
- getNodeValue, B-26
- getNotificationProperty, B-58
- getNumberOfTaskApprovals, B-59
- getPreference, B-28
- getPreviousTaskApprover, B-59
- getProcessId, B-28
- getProcessOwnerId, B-28
- getProcessURL, B-28
- getProcessVersion, B-29
- getReportees, B-64
- getTaskAttachmentByIndex, B-59
- getTaskAttachmentByName, B-60
- getTaskAttachmentContents, B-60
- getTaskAttachmentsCount, B-60
- getTaskResourceBindingString, B-61
- getUserAliasId, B-29
- getUserProperty, 16-13, B-64
- getUserRoles, B-65
- getUsersInGroup, B-66
- getVariableData, 16-13, B-41
- getVariableProperty, B-41
- hours-from-dateTime, B-5
- implicit-timezone, B-5
- in transformations, 37-19
- index-within-string, B-11
- integer, B-30
- isUserInRole, B-66
- last-index-within-string, B-12
- left-trim, B-12
- listUsers, B-30
- location of function descriptions, 6-5
- lookupGroup, B-67
- lookup-table, B-1
- lookupUser, B-31, B-67
- lookup-xml, B-56
- lower-case, B-13
- matches, B-13
- max-value-among-nodeset, B-44
- mediator XPath extension, B-47
- minutes-from-dateTime, B-6
- min-value-among-nodeset, B-45
- month-from-dateTime, B-6
- parseEscapedXML, 6-51, B-32
- position, 6-45
- prefixed with xp20 or orcl, 37-19
- processXQuery, B-32
- processXSLT, 16-8, B-33
- processXSQL, B-37
- query-database, B-2
- readBinaryFromFile, B-37
- readFile, B-38
- right-trim, B-13
- search, B-38
- seconds-from-dateTime, B-6
- selecting an data sequence element, 6-45
- sequence-next-val, B-2
- SOA XPath extension, B-1
- square-root, B-45
- subtract-dayTimeDuration-from-dateTime, B-6
- timezone-from-dateTime, B-7
- translateFromNative, B-45
- translateToNative, B-46
- upper-case, B-14
- wfDynamicGroupAssign, B-61

- wfDynamicUserAssign, B-62
- workflow service, B-58
- writeBinaryToFile, B-40
- year-from-dateTime, B-7
- Fusion Order Demo
 - deploying, 3-12
 - deploying in a partition, 3-22
 - installing schema, 3-15
 - integration with spring, 49-21
 - introduction, 3-1
 - running, 3-23
 - setting up, 3-3
 - Store Front module, 3-1
 - WebLogic Fusion Order Demo, 3-2
 - introduction, 3-2
- Fusion Web Application (ADF) application
 - template, 32-2
- FusionOrderDemo_R1PS3.zip, 3-3
- FYI assignee
 - configuring, 27-38
 - definition, 26-5, 27-38
 - must first claim an FYI task before dismissing it, 28-22
 - tasks are not actionable, 27-66
 - workflow participant type, 26-5, 27-38

G

- genEmptyElem function
 - description, 6-50, B-43
- generateGUID function
 - description, B-21
- generate-guid function
 - description, B-53
- getChildElement function
 - description, B-44
- getContentAsString function
 - description, B-23
- get-content-as-string function
 - description, B-10
- getConversationId function
 - description, B-23
- getCreator function
 - description, B-24
- getCurrentDate function
 - description, 6-20, B-24
- getCurrentDateTime function
 - description, 6-20, B-24
- getCurrentTime function
 - description, 6-20, B-24
- getDefaultRealmName function
 - description, B-63
- getDomainId function
 - description, B-25
- getElement function
 - description, B-25
- getFaultAsString function
 - description, 11-25
- getGroupIdsFromGroupAlias function
 - description, B-26

- getGroupProperty function
 - description, B-63
- getInstanceId function
 - description, B-26
- getLinkStatus function
 - description, B-40
- get-localized-string function
 - description, B-11
- getManager function
 - description, B-63
- getMessage function
 - description, B-44
- getNodes function
 - description, B-27
- getNodeValue function
 - description, B-26
- getNotificationProperty function
 - description, B-58
- getNumberOfTaskApprovals function
 - description, B-59
- getPreference function
 - description, B-28
- getPreviousTaskApprover function
 - description, B-59
- getProcessId function
 - description, B-28
- getProcessOwnerId function
 - description, B-28
- getProcessURL function
 - description, B-28
- getProcessVersion function
 - description, B-29
- getReportees function
 - description, B-64
- getTaskAttachmentByIndex function
 - description, B-59
- getTaskAttachmentByName function
 - description, B-60
- getTaskAttachmentContents function
 - description, B-60
- getTaskAttachmentsCount function
 - description, B-60
- getTaskResourceBindingString function
 - description, B-61
- getUserAliasId function
 - description, B-29
- getUserProperty function
 - description, B-64
 - example, 16-13
- getUserRoles function
 - description, B-65
- getUsersInGroup function
 - description, B-66
- getVariableData function
 - description, 6-18, B-41
 - example, 16-13
 - throws selectionFailure if result node set size is greater than one, B-41
 - using in mathematical calculations, 6-17
- getVariableProperty function

- description, B-41
- global task variable name
 - specifying in human task activities, 27-84
- globalTxMaxRetry property
 - description, C-2
- globalTxRetryInterval property
 - description, C-2
- governance
 - Oracle Enterprise Repository, A-55
- group names
 - case sensitive by default, 27-12
- group vote
 - configuring, 27-31
 - consensus percentage, 27-33
 - immediately triggering a voted outcome when a minimum percentage is met, 27-34
 - specifying group voting details, 27-33
 - waiting until all votes are in before triggering an outcome, 27-34

H

- headers
 - normalized message header properties, H-1
 - SOAP headers, 6-53
- Headers tab
 - in activities, A-5
- heap size
 - increasing, 37-49, 42-12
- History window
 - location of in Oracle JDeveloper, 4-5
- hours-from-dateTime function
 - description, B-5
- HTTP binding
 - binding component, 2-11, 2-17
 - capabilities, A-43
 - configuring with the HTTP Binding Wizard, 34-7
 - creating your own schema, 34-8
 - enabling basic authentication, 34-8
 - in SOA composite applications, 34-5
 - limitations in SOA composite applications, 34-5
 - support for HTTPS in inbound and outbound directions, 34-7
 - supported inbound and outbound interactions, 34-5
 - supported operation types, 34-7
 - supported XSD types, 34-6
 - unsupported HTTP headers, 34-6
- HTTP headers
 - unsupported, 34-6
- human task
 - service component, 2-7
- human task activity
 - associating with a BPEL process, 27-78
 - identification key, 27-85
 - including the task history of other tasks, 27-85
 - scope name and global task variable name, 27-84
 - specifying a task initiator and task priority, 27-81
 - specifying a task title, 27-80
 - specifying task parameters, 27-81

- task owner, 27-85
- viewing BPEL callbacks, 27-87
- human task definition
 - associating with a BPEL process, 27-2
- Human Task Editor
 - abruptly completing a condition, 27-42
 - accessing the sections of, 27-6
 - actionable emails, 31-32
 - allowing all participants to invite other participants, 27-42
 - assigning task participants by name or expression, 27-26, 27-53
 - bypassing task participants, 27-31, 27-35, 27-38
 - changing character set encoding, 27-65
 - configuring the error assignee, 27-51
 - creating a human task, 27-3
 - customizing notification headers, 27-67
 - editing notification messages, 27-64
 - escalate after policy, 27-58
 - escalating, renewing, or ending a task, 27-55
 - escalation and expiration policy overview, 27-56
 - escalation rules, 27-59
 - expire after policy, 27-57
 - FYI assignee task participant, 27-38
 - group voting details, 27-33
 - inviting additional task participants, 27-31, 27-35, 27-37
 - making email messages actionable, 27-66
 - multilingual settings, 27-54, 31-31
 - never expire policy, 27-57
 - notification preferences, 27-60
 - notifying recipients of changes to task status, 27-62
 - parallel task participant, 27-31
 - renew after policy, 27-58
 - securing notifications, 27-65, 31-34
 - sending email notifications to groups and application roles, 27-66
 - sending task attachments with email notifications, 27-66
 - serial task participant, 27-35
 - setting up reminders, 27-65
 - sharing attachments and comments with task participants, 27-34
 - showing the Oracle BPM Worklist URL in notifications, 27-65
 - single approver task participant, 27-23
 - specifying access policies, 27-67
 - specifying business event callbacks, 27-75
 - specifying callback classes, 27-74
 - specifying digital signatures, 27-71
 - task attachments with email notifications, 31-34
 - task category, 27-11
 - task outcome, 27-9
 - task owner specification through the user directory, 27-12
 - task owner specification through XPath expressions, 27-16
 - task participants, 27-20
 - task payload data structure, 27-17

- task priority, 27-11
- task routing and customization in BPEL
 - callbacks, 27-77
- task title, 27-8
- time limits for acting on tasks, 27-30, 27-35, 27-37
- WordML style sheets in attachments, 27-54
- human tasks
 - creating, 27-3
 - designing a human task, 26-13
- human workflow
 - access rules, 26-9
 - application roles, 26-5
 - case sensitivity, 31-47
 - concepts, 26-3
 - database views, 31-60
 - definition, 26-1
 - groups, 26-5
 - integration with Oracle WebLogic Server, 31-48
 - participant assignments, 26-5
 - participant types, 26-4
 - participants, 26-4
 - routing policies, 27-39
 - System MBean Browser properties, K-8
 - task assignments, 26-6
 - task deadlines, 26-7
 - task stakeholders, 26-7
 - use cases, 26-11
 - users, 26-5

I

- ICommand
 - clear, G-3
 - command line, 58-5
 - delete, G-3
 - detailed command descriptions, G-3
 - export, G-5
 - sample, G-18
 - general command and option syntax, 58-2
 - import, G-10
 - log, G-17
 - operations, G-1
 - regular expressions, G-18
 - remote execution, 58-6
 - rename, G-14
 - running, 58-1
 - sample export file, G-18
 - summary of commands, G-1
 - syntax, 58-2
 - syntax, object names, 58-3
 - XML file, G-15
- ICommand utility, 58-1
- ICommand web service, 56-4
- idempotent property
 - description, C-3
- identification key
 - specifying in human task activities, 27-85
- identity service
 - definition, 26-28, 31-11
 - determining a user's local language and time

- zone, 29-72
- Enterprise JavaBeans, SOAP, and Java
 - support, 31-2
- functions
 - getDefaultRealmName, B-63
 - getGroupProperty, B-63
 - getManager, B-63
 - getReportees, B-64
 - getUserProperty, B-64
 - getUserRoles, B-65
 - getUsersInGroup, B-66
 - isUserInRole, B-66
 - lookupGroup, B-67
 - lookupUser, B-67
- providers, 31-12, 31-13
- support for in workflows, 31-11
- supported task operations, 31-11
- use with JAZN, 31-11, 31-12
- use with LDAP, 31-11, 31-12
- WSDL file location, 31-2
- if activity
 - capabilities, A-19
 - defining conditional branching, 10-5
 - replaces the switch activity in BPEL 2.0, A-3
- ignoreMissingFromData attribute
 - selecting in an assign activity, A-9
 - using, 6-34
- IM activity
 - capabilities, A-20
 - notifications support, 16-9
- implicit-timezone function
 - description, B-5
- import
 - source and target schemas into a transformation, 37-9
- indexes
 - in data objects, 52-17
- indexing methods
 - using XPath, 6-46
- index-within-string function
 - description, B-11
- inline variables initialization
 - in BPEL 2.0, 8-5
- inMemoryOptimization property
 - description, C-2
- insertMissingToData attribute
 - selecting in an assign activity, A-9
 - using, 6-34
- instances
 - starting new, 8-8
- integer function
 - description, B-30
- integration
 - of Java and WSDL-based components in the same composite, 35-1, 49-2, 49-14
- interaction patterns
 - asynchronous interaction with a notification timer, 5-5
 - asynchronous interaction with a timeout, 5-4
 - asynchronous interactions, 5-3

- common patterns between a BPEL process and another application, 5-1, 23-1
- multiple interactions, 5-10
- of interaction between a BPEL process and another application, 5-1, 23-1
- one request, a mandatory response, and an optional response, 5-8
- one request, multiple responses, 5-6
- one request, one of two possible responses, 5-7
- one-way message, 5-1
- partial processing, 5-9
- synchronous interactions, 5-2
- Invalid Settings error message, A-55
- invoke activity
 - adding a conversation ID, 8-10
 - adding to an asynchronous service, 8-3
 - capabilities, A-20
 - definition, 4-7, 7-1
 - in asynchronous services, 8-3, 8-8
 - in synchronous services, 7-1, 7-5
 - throwing faults with assertion conditions, 11-45
- isUserInRole function
 - description, B-66

J

- JAR
 - See* .JAR Files
- .JAR files
 - adding custom classes and JAR files, 13-6
 - adf-desktop-integration.jar, 32-2
 - creating a JAR file for deployment, 40-18
 - resourcebundle.jar file, 32-2
 - wsclient.jar, 32-2
- Java
 - support in workflow services, 31-2
- Java applications
 - wrapped as SOAP services, 13-1
- Java Connector Architecture (JCA)
 - definition, 1-3
- Java embedding
 - bpelx:exec extension, 13-4
 - example, 13-7
 - in a BPEL process, 13-1
 - using thread.sleep(), 13-8
- Java embedding activity
 - capabilities, A-21
 - using Java embedding in a BPEL process, 13-7
- Java interfaces
 - creating Java interface integration with SOA composite applications, 35-11
 - integrating Enterprise JavaBeans and SOA composite applications, 35-1, 35-2
 - integration of Java and WSDL-based components in the same SOA composite application, 49-2
 - selecting when creating a partner link, 4-8
 - using with spring service components, 49-2
- JAXB
 - configuring the workflow client, 31-53
 - support in SOA composite applications, 49-28

- JAZN
 - storing a user's local language and time zone, 29-72
 - use with identity service, 31-11, 31-12
- jdbc.port parameter, 3-15
- jdbc.sid parameter, 3-15
- jdbc.urlBase parameter, 3-15
- jdeveloper.home parameter, 3-15
- JMS
 - definition, 1-3
- JMS adapter
 - capabilities, A-44
 - definition, 34-10
 - sensor publish type, 17-2
- JMS queue
 - sensor publish type, 17-2
- JMS topic
 - sensor publish type, 17-2
- join conditions
 - using in target activities, 9-11

K

- keepGlobalVariables property
 - description, C-2
- keepSrcElementName attribute
 - selecting in an assign activity, A-9
 - using, 6-35
- knowledge module
 - Oracle BAM, 54-2

L

- languages
 - changing
 - from jazn xml file, 29-74
 - preferences, 29-72
 - setting in JAZN, 29-72
 - setting in LDAP, 29-72
- large documents
 - best practices for handling, 42-1
 - importing large data sets in Oracle B2B, 42-20
 - large numbers of mediators in composites, 42-20
 - limitations on concurrent processing, 42-12
 - opaque schema for processing large payloads, 42-12
 - processing in Oracle B2B, 42-16
 - setting a default JTA timeout for large documents, 42-12
 - setting audit levels, 42-13
 - use cases for handling, 42-1
 - using a flow with multiple sequences, 42-19
 - using a flow with no sequence, 42-20
 - using a flow with one sequence, 42-19
 - using assign activities in BPEL and mediator, 42-13
 - using large numbers of activities in BPEL processes (with FlowN), 42-19
 - using large numbers of activities in BPEL processes (without FlowN), 42-19

- using XSLT transformations for repeating structures, 42-15
 - using XSLT transformations on large payloads (for BPEL and mediator), 42-14
- large XML documents
 - processing with complex structures, 42-11
 - processing with repeating constructs, 42-10
- last-index-within-string function
 - description, B-12
- layouts, data object, 52-9
- LDAP
 - storing a user's local language and time zone, 29-72
 - used with identity service, 31-11, 31-12
- left-trim function
 - description, B-12
- listUsers function
 - description, B-30
- literal strings
 - assigning, 6-18
- literal XML
 - variable initialization, 6-13
- localization, worklist, 29-72
- Log window
 - location of in Oracle JDeveloper, 4-5
- looking up cross reference tables, 46-15
 - xref
 - lookupXRef function, 46-15
- lookup fields, 52-4
- lookupGroup function
 - description, B-67
- lookup-table function
 - description, B-1
- lookupUser function
 - description, B-31, B-67
- lookupValue functions
 - dvm
 - lookupValue function, 44-8
 - lookupValueIM function, 44-9
- lookup-xml function
 - description, B-56
- lower-case function
 - description, B-13

M

- managed.server parameter, 3-22
- managed.server.port parameter, 3-22
- management chains
 - definition, 27-24
 - participant lists, 27-26
 - rule-based, 27-24
- ManualRuleFire web service, 56-4
- map parameters
 - creating in transformations, 37-37
- map variables
 - creating in transformations, 37-37
- mapped attributes, 29-61, 29-62
 - using, 29-61
 - values, 31-17

- master and detail processes
 - creating, 15-7
 - definition, 15-1
 - receive signal activity, A-28
 - signal activity, A-34
- matches function
 - description, B-13
- maxOccurs attribute, 6-45, 6-46
 - setting for transformations, 37-50
- max-value-among-nodeset function
 - description, B-44
- mediator
 - mediator files, 18-4
 - service component, 2-7
- mediator creation
 - specifying operation or event subscription properties, 18-33
- mediator files
 - .componentType, 18-4
 - composite.xml, 18-4
 - .mplan, 18-4
 - .wsdl, 18-4
- mediator XPath extension functions, B-47
- message filtering, 60-7, 61-11, 62-12
- message schemas
 - updating, 2-15
 - viewing, 2-15
- message source advanced formatting, 53-10
- message sources, 53-1
- message types
 - support for simple types in a message part, 2-14
- MessageFilter, 60-7, 61-11, 62-12
- MessageFilterFactory, 60-7, 61-11, 62-12
- messages
 - receiving, 60-6, 61-9, 62-9
 - rejecting, 60-7, 61-11, 62-12
- MessagingClientFactory, 60-3
- MessagingClient.receive, 60-7, 61-9, 62-10
- MessagingClient.registerAccessPoint, 60-6, 61-9, 62-10
- MessagingClient.registerMessageFilter, 60-7, 61-11, 62-12
- metadata
 - service components, 24-13
- Metadata Service (MDS)
 - creating a SOA-MDS connections, 40-35
 - definition, 1-7
- MIME
 - creating composites that use MIME attachments, 42-4
- MIME mapping
 - Excel workbook, 32-3
- MinBPELWait property, 14-12
- minimum wait time
 - MinBPELWait property, 14-12
- minOccurs attribute
 - setting for transformations, 37-50
- minutes-from-dateTime function
 - description, B-6
- min-value-among-nodeset function

- description, B-45
- modes
 - xref
 - populateXRefRow function, 46-11
 - populateXRefRow1M function, 46-13
- modifying a mediator, 18-33
 - modifying event subscriptions, 18-34
 - modifying operations, 18-34
- modifying cross reference tables
 - adding a column, 46-8
- modifying mediator event subscriptions, 18-34
- modifying mediator operations, 18-34
- month-from-dateTime function
 - description, B-6
- MQ adapter
 - capabilities, A-44
 - definition, 34-10
- MTOM
 - adding MTOM attachments to web services, 42-9
 - using SOAP, 42-2
- multilingual settings
 - specifying in tasks, 27-54, 31-31
- multipart WSDLs
 - adding to a composite, 2-14
- myRole attribute
 - definition, 8-7

N

- named templates
 - creating, 37-21
 - in functions, 37-21
- names and expressions
 - definition, 27-24
 - participant list, 27-26
 - rule-based, 27-24
- namespace prefix, B-40
- namespaces
 - BPEL 1.1 prefix, 6-4, B-40
 - BPEL 2.0 prefix, 6-4, B-40
 - declaring extension namespaces in BPEL 2.0, 6-55
 - ensuring the uniqueness of WSDL
 - namespaces, 2-15
- naming conventions
 - for BPEL projects, 4-2
- nonBlockingInvoke property
 - description, C-3
- normalized message header properties
 - Oracle BPEL Process Manager, H-2
 - Oracle Web Services Addressing, H-3
- NOT operator, 52-5
- notification messages
 - editing, 27-64
- notification services
 - actionable emails, 31-32
 - configuring the notification channel, 31-30
 - custom notification headers, 31-36
 - definition, 26-28
 - error message support, 31-29
 - multilingual settings, 31-31

- notification contents, 31-28
- reliability support, 31-29
- sending inbound and outbound
 - attachments, 31-34
- sending inbound comments, 31-34
- sending reminders, 31-34
- sending secure notifications, 31-34
- setting automatic replies to unprocessed
 - messages, 31-35
- specifying participant notification
 - preferences, 27-60
- notifications
 - allowing the end user to select the notification
 - channels, 16-14
 - configuring in Oracle JDeveloper, 16-3
 - customizing notification headers, 27-67
 - definition, 26-8
 - dynamically setting email addresses and telephone
 - numbers, 16-12
 - email attachment support, 16-7
 - email support, 16-2, 16-4
 - formatting the email message body as
 - HTML, 16-8
 - IM support, 16-9
 - making email messages actionable, 27-66
 - securing to exclude details, 27-65
 - selecting recipients by browsing the user
 - directory, 16-13
 - sending email notifications to groups and
 - application roles, 27-66
 - sending task attachments with email
 - notifications, 27-66
 - setting up, 16-3
 - showing the Oracle BPM Worklist URL, 27-65
 - SMS support, 16-10
 - voice mail support, 16-12
- notifications and reminders
 - in tasks, 31-27
- numeric values
 - assigning, 6-17

O

- onAlarm branch
 - of pick activities, 14-2, A-25
 - of scope activities, 14-3, A-26
- one-to-many mapping, 44-4
- onEvent branch
 - creating in a scope activity, 14-15
 - specifying events to wait for message
 - arrival, 14-13
- one-way invocations
 - introduction, 12-4
- oneWayDeliveryPolicy property
 - description, C-2
 - setting, 12-4
- one.way.returns.fault
 - property, K-4
- onMessage branch
 - of pick activities, 14-2, A-25

- of scope activities, 14-3, A-26
 - simultaneous onMessage branches in BPEL 2.0, 14-6
- operators
 - AND operator, 52-5
- optimization
 - streaming attachments, 42-6
- OR operator, 52-5
- Oracle Application Development Framework (ADF)
 - binding component, 1-5
- Oracle Applications adapter
 - capabilities, A-44
 - definition, 34-10
- Oracle B2B
 - attachments, 42-9
 - binding component, 2-11, 2-17
 - capabilities, A-42
 - definition, 1-5, 34-11
 - properties, K-4
 - streaming, 42-12
- Oracle BAM, 50-29
 - definition, 1-5, 34-11
 - See Oracle Business Activity Monitoring Server connection, 51-2
- Oracle BAM Adapter, 50-1
- Oracle BAM knowledge modules, 54-2
- Oracle BAM Server
 - creating a BPEL sensor, 50-29
 - creating a BPEL sensor action, 50-30
 - creating a connection to, 50-25
- Oracle BAM Server connection, 50-25
- Oracle BPEL Designer
 - layout, 4-3
- Oracle BPEL Process Manager
 - System MBean Browser properties, K-8
- Oracle BPM Worklist
 - See worklist
- Oracle Business Activity Monitoring
 - capabilities, A-42
 - creating a BPEL sensor action for Oracle BAM Server, 50-30
 - creating a BPEL sensor for Oracle BAM Server, 50-29
 - creating a connection to Oracle BAM Server, 50-25
 - definition
 - integration with Oracle BPEL Process Manager sensors, 50-29
 - message batching limitations, 50-32
 - overview, 50-29
- Oracle Enterprise Manager Fusion Middleware Control
 - improving the loading of pages, 42-20
 - properties, K-4
- Oracle Enterprise Repository
 - design-time governance, A-55
- Oracle Internet Directory
 - storing a user's local language and time zone, 29-72
- Oracle JDeveloper
 - adapters, 4-13
 - configuring notifications, 16-3
 - creating sensors, 17-3
 - installing the Oracle SOA Suite extension, 2-1
 - location of Application Navigator, 4-4
 - location of Component Palette, 4-4
 - location of Designer window, 4-4
 - location of History window, 4-5
 - location of Log window, 4-5
 - location of Property Inspector, 4-5
 - location of Source window, 4-5
 - location of Structure window, 4-5
 - overview of rules designer environment, 24-5
 - transformations, 37-7
- Oracle JDeveloper project
 - desktop integration, adding, 32-2
- Oracle Mediator
 - define routing rules, 19-1
 - definition, 18-1
 - routing rules, 19-1
 - System MBean Browser properties, K-8
- Oracle Mediator component creation
 - mediator files, 18-4
- Oracle Mediator Editor, 18-4
 - environment
 - Application Navigator, 18-4
 - History Window, 18-5
 - Log Window, 18-5
 - Oracle Mediator Editor, 18-4
 - Property Inspector, 18-5
 - Source View, 18-5
 - Structure Window, 18-5
 - layout, 18-4
- Oracle Mediator error handling
 - actions, 21-4
 - conditions, 21-2
 - fault bindings, 21-8
 - fault policy, 21-1
 - introduction, 21-1
 - using, 21-10
 - XML schema files, 21-11
- Oracle Service Bus (OSB)
 - invocation by the direct binding service, 34-12, 36-6, A-43
- Oracle Service Registry
 - changing endpoint locations in the registry control, A-52
 - configuring a SOA project to invoke a service from the registry, A-46
 - configuring the inquiry URL, UDDI service key, and endpoint address for runtime, A-51
 - creating a connection to, A-45
 - dynamically resolving the SOAP endpoint location, A-47
 - dynamically resolving the WSDL endpoint location, A-48
 - publishing a business service, A-45
 - publishing and browsing, A-44
 - publishing WSDLs from multiple SOA partitions, A-54

- publishing WSDLs to UDDI for multiple partitions, A-54
- System MBean Browser properties, K-9
- Oracle SOA Suite
 - definition, 1-2
- Oracle User Messaging Service (UMS)
 - configuring, 59-1
 - definition, 16-2
- oracle.composite.faultBindingFile
 - property, 11-7, K-3
- oracle.composite.faultPolicyFile
 - property, 11-7, K-3
- oracle.home parameter, 3-22
- oracle.webservices.local.optimization
 - property, K-3
 - streaming attachments, 42-6
- OrderApprovalHumanTask project, 3-7
- OrderBookingComposite composite
 - business rules, used in, 3-10
- OrderBookingComposite project, 3-7
 - flow described, 3-8
- OrderProcessor BPEL process, 3-8
- OrderSDOComposite project, 3-7
- organizing data objects, 52-10
- org.quartz.scheduler.idleWaitTime
 - properties, K-4
- overview, 17-2
- OXM
 - support in SOA composite applications, 49-28

P

- packaging
 - of artifact files for deployment, 40-2
- pages
 - improving the loading of pages in Oracle Enterprise Manager Fusion Middleware Control, 42-20
- parallel
 - definition, 27-31
 - workflow participant type, 27-31
- parallel blocks
 - definition, 27-21
- parallel branches
 - customizing the number, 9-11
- parallel flows
 - definition, 9-1
- parallel participant types
 - specifying where to store the subtask payload, 27-53
- parseEscapedXML function
 - description, 6-51, B-32
- partial processing
 - BPEL process as the client, 5-10
 - BPEL process as the service, 5-10
 - definition, 5-9
- participant assignments
 - definition, 26-5
- participant lists
 - rulesets, 27-28
- value-based management chains, 27-26
- value-based names and expressions, 27-26
- participant types
 - FYI assignee, 26-4, 26-5, 27-38
 - parallel, 26-4, 27-31
 - serial, 26-4, 27-35
 - single approver, 26-4, 27-23
- partitions
 - ant scripts, 40-55
 - creating, 40-24
 - default partition, 40-24
 - deployment in, 40-24
 - in the Fusion Order Demo, 3-22
 - issues with deploying the same composite with a human workflow into multiple partitions, 40-24, 40-46
 - selecting a partition during deployment, 40-24
- partner link activity
 - capabilities, A-22
- partner links
 - adding to an asynchronous service, 8-2
 - creating, 4-9
 - definition, 4-7
 - for an inbound adapter, 4-11
 - for an outbound adapter, 4-10
 - from an abstract WSDL to call a service, 4-11
 - from an abstract WSDL to implement a service, 4-11
 - from an existing human task, business rule, or Oracle Mediator, 4-12
 - in asynchronous services, 8-2, 8-6, 8-7
 - in synchronous services, 7-1
 - Oracle BAM, 50-27
 - overview, 4-7
 - specifying a WSDL file, 4-8
 - using a dynamic partner link at runtime, 8-11
 - with human tasks or business rules, 4-12
- partnerLinkType
 - definition, 8-6
- partnerRole attribute
 - definition, 8-7
- PartnerSupplierComposite project, 3-7
- passThroughHeader
 - property, K-3
- permissions
 - copying, 52-7
 - data objects, 52-6
 - setting on folders, 52-11
- phase activity
 - BPEL scope creation, 48-4
 - business rule service component creation, 48-5
 - capabilities, A-23
 - mediator service component creation, 48-4
 - using with two-layer business process management, 48-3
- pick activity
 - adding correlations on an OnMessage branch, A-26
 - capabilities, A-24
 - code example, 14-5

- condition branches, 14-2
- creating, 14-3
- differences with a receive activity, 14-3
- for timeouts, 14-1
- onAlarm branch, 14-2, A-26
- onMessage branch, 14-2, A-25
- simultaneous onMessage branches in BPEL 2.0, 14-6
- throwing faults with assertion conditions, 11-45
- policies
 - adding security policies, 2-24
 - attaching, 39-2
 - definition, 39-1
 - overriding client property values, 39-6
 - overriding policy configuration property values, 39-6
 - overriding server property values, 39-8
 - supported categories, 39-1
- populating cross reference tables, 46-9
 - xref
 - populateXRefRow1M function, 46-12
- portlets
 - See task list portlets
- ports
 - in synchronous services, 7-1
- portType
 - definition, 8-6
- position function
 - description, 6-45
- process definitions
 - importing in BPEL 2.0, 6-44
- process initiation
 - in BPEL test suites, 41-3
- processes
 - naming conventions, 4-2
- processXQuery function
 - description, B-32
- processXSLT function
 - description, B-33
 - example, 16-8
- processXSQL function
 - description, B-37
- projects
 - naming conventions, 4-2
 - ViewController, 32-2
- properties
 - adapter rejected messages, K-4
 - completionPersistPolicy, C-2, K-1
 - composite.xml file properties, K-3
 - cross references, K-6
 - deployment descriptors overview, K-1
 - disableAsserts, C-2, K-1
 - endpointURI, K-3
 - fault policy, K-4
 - globalTxMaxRetry, C-2, K-1
 - globalTxRetryInterval, C-2, K-1
 - human workflow notifications, K-5
 - human workflow System MBean Browser, K-8
 - human workflow task service, K-5
 - idempotent, C-3, K-2

- inMemoryOptimization, C-2, K-1
- JCA adapter normalized message header
 - properties overview, K-2
- keepGlobalVariables, C-2, K-1
- nonBlockingInvoke, C-3, K-2
- normalized message header properties
 - overview, K-2
- normalized message properties, H-1
- oneWayDeliveryPolicy, 12-4, C-2, K-1
- one.way.returns.fault, K-4
- Oracle B2B, K-4, K-6
- Oracle B2B normalized message header properties
 - overview, K-3
- Oracle BPEL Process Manager, K-5
- Oracle BPEL Process Manager normalized message
 - header properties overview, K-2
- Oracle BPEL Process Manager System MBean
 - Browser, K-8
- Oracle Enterprise Manager Fusion Middleware
 - Control, K-4
- Oracle Mediator, K-6
- Oracle Mediator System MBean Browser, K-8
- Oracle Service Registry, K-9
- Oracle Web Services Addressing normalized
 - message header properties overview, K-2
- oracle.composite.faultBindingFile, 11-7, K-3
- oracle.composite.faultPolicyFile, 11-7, K-3
- oracle.webservices, K-3
- org.quartz.scheduler.idleWaitTime, K-4
- passThroughHeader, K-3
- retryCount, K-4
- retryInterval, K-4
- rolesAllowed, K-3
- sensorActionLocation, C-2, K-1
- sensorLocation, C-2, K-1
- service and reference binding components, K-6
- SOA Infrastructure, K-5
- SOA Infrastructure System MBean Browser, K-7
- streamIncomingAttachments, K-3
- streamOutgoingAttachments, K-3
- System MBean Browser, K-7
- transaction, 12-1, 12-2, 12-3, C-2, K-1
- uddiCacheLifetime, K-9
- validateXML, C-3, K-2
- Properties tab
 - in activities, A-5
- property aliases
 - creating for correlation sets, 8-23
- Property Inspector
 - location of in Oracle JDeveloper, 4-5
- public views
 - sensors, D-1
- publish types
 - creating a custom publisher, 17-12
 - custom, 17-2
 - database, 17-2
 - definition, 17-2
 - JMS Adapter, 17-2
 - JMS queue, 17-2
 - JMS topic, 17-2

purge script
 deleting instances and rejected messages, 42-20

Q

Qname
 fault name, 11-5
qualifier, 44-2
 qualifier order, 44-3
qualifier order, 44-3
query-database function
 description, B-2

R

readBinaryFromFile function
 description, B-37
readFile function
 description, B-38
 limitation on web server file access requiring
 authorization, B-38
 reading files from absolute directory paths, B-38
receive activity
 adding to an asynchronous service, 8-4
 associating with correlation sets, 8-22
 capabilities, A-27
 create instance, 8-8
 creating new instances, 8-8
 differences with a pick activity, 14-3
 in asynchronous services, 8-4, 8-8
 setting timeouts for request-response
 operations, 14-7
 throwing faults with assertion conditions, 11-45
receive signal activity
 capabilities, A-28
 in master and detail processes, 15-8
receiving a message, 60-6, 61-9, 62-9
references
 adding, 2-16, 2-18
 definition, 1-5, 1-7, 2-12
 deleting, 2-18
 wiring, 2-21
regular expressions
 ICommand, G-18
rejecting messages, 60-7, 61-11, 62-12
reminders
 for task notifications, 31-34
remoteFault
 definition, 11-6
remove entity activity
 capabilities, A-28
renaming data objects, 52-17
renaming folders, 52-12
renewing
 tasks, 27-55
repeating constructs
 processing large XML, 42-10
repeating elements
 in transformations, 37-28
repeatUntil activity
 capabilities, A-29
 defining conditional branching, 10-10
replay activity
 capabilities, A-30
 creating, 11-37
 re-executing activities in a scope activity, 11-37
replayFault
 definition, 11-6
reply activity
 capabilities, A-30
reporting schema
 for database publish type of sensors, D-1
reports
 correcting memory errors when generating for
 transformations, 37-49
 customizing sample XML generation for
 transformations, 37-50
 generating for transformations, 37-48
 worklist, 29-67
resequencing
 BestEffort resequencer, 22-5
 configuring, 22-8
 configuring the strategy, 22-9
 definition, 22-1
 determining the level, 22-8
 FIFO resequencer, 22-4
 groups and sequence IDs, 22-1
 identification of groups and sequence IDs, 22-2
 limitations, 22-12
 order types, 22-2
 standard resequencer, 22-3
resource bundles, 31-44
 class loading, 31-44
 for displaying tasks in different languages, 27-54,
 31-31
 specifying stage and participant names, 31-47
Resource Palette
 introduction, 2-5
 using, 2-13
resourcebundle.jar file, 32-2
rethrow activity
 capabilities, A-31
 rethrowing faults, 11-26
 supported in BPEL 2.0 projects, A-31
retryCount
 properties, K-4
retryInterval
 properties, K-4
revisions
 activating, 2-27
 invoking the default revision, 2-19
 retiring, 2-27
 setting the default revision, 2-28
 turning off, 2-27
 turning on, 2-27
 undeploying, 2-28
right-trim function
 description, B-13
roles
 for partner links in asynchronous services, 8-6

- rolesAllowed
 - property, K-3
- routing policies
 - available types, 27-40
 - business rules, 27-41
 - completing parent subtasks of early completing subtasks, 27-44
 - enabling early completion in parallel subtasks, 27-43
 - external routing, 27-41, 27-49
 - routing a task to all participants in the order specified, 27-40
 - selecting, 27-39
- routing rules, 19-1
 - define, 19-1
 - defining, 19-1
 - filter expression, 19-14
 - introduction, 19-1
- routing slip
 - definition, 27-30
- RPC styles
 - differences with document-literal styles in WSDL files, 6-1, 6-52
- rulesets
 - management chains, 27-24
 - names and expressions, 27-24
 - participant lists, 27-28
- runtime config service
 - definition, 26-28
 - Enterprise JavaBeans, SOAP, and Java support, 31-2
 - supported task operations, 31-16
 - WSDL file location, 31-3
- runtime exceptions, 11-5
- runtime faults
 - definition, 11-5
 - example, 11-24
- RuntimeFault.wsdl file
 - importing into a process, 11-24

S

- samples
 - business events, 38-1
 - business rules, 24-23
 - domain value maps, 44-22
 - dynamic assignment functions, 31-39
 - email notifications, 28-32
 - Hello World, 7-1
 - human workflow, 26-26, 27-48, 27-77
 - internationalization of attribute labels, 31-19
 - iterative design, 27-48
 - mediator asynchronous response, 19-58
 - mediator routing messages, 19-47
 - notifications, 16-8
 - Oracle BPEL Process Manager, 6-2
 - Oracle SOA Suite, 1-9
 - transformations, 37-22
 - two-layer business process management, 48-7
 - workflow event callbacks, 27-77

- SAR file
 - definition, 1-8, 40-2
 - deploying, 40-16
- SCA *See* Service Component Architecture
- sca-build.properties file, 3-20
- schema files
 - creating a transformation map file from imported schemas, 37-9
 - replacing in the XSLT Mapper, 37-41
- schemas
 - updating message schemas, 2-15
 - viewing message schemas, 2-15
- scope activity
 - adding descriptive notes and images, 11-30
 - capabilities, A-32
 - creating, 11-31
 - creating an onEvent branch, 14-14
 - fault handling, 11-29
 - re-executing with a replay activity, 11-37
 - using a fault handler in a scope activity, 11-33
- scope name
 - specifying in human task activities, 27-84
- SDO
 - See* Service Data Objects (SDO)
- search function
 - description, B-38
- seconds-from-dateTime function
 - description, B-6
- security filters
 - copying, 52-15
 - on data objects, 52-13
- security model
 - for workflow services, 31-4
 - in SOAP web services, 31-5
 - workflow context on behalf of a user, 31-5
- security policies
 - See* policies
- seed.bam.do parameter, 3-21
- seedBAMServerObjects ant script, 3-23
- seedDemoUsers ant script, 3-23
- seedFodJmsResources ant script, 3-23
- sensor actions
 - configuring, 17-8
 - creating a BPEL sensor action for Oracle BAM Server monitoring, 50-30
 - viewing metadata, 17-15
 - XSD schema file, D-5
- sensor data
 - persisting in a reporting schema, D-1
- sensorActionLocation property
 - description, C-2
- sensorLocation property
 - description, C-2
- sensors, 17-2, 50-29
 - activity sensors, 17-2
 - BPEL reporting schema, D-1
 - configuring, 17-4
 - creating a BPEL sensor for Oracle BAM Server to monitor, 50-29
 - creating a connection to Oracle BAM

- Server, 50-25
 - creating a custom publish type, 17-12
 - creating in Oracle JDeveloper, 17-3
 - definition, 17-2
 - evaluation time, 17-5
 - fault sensors, 17-2
 - integration with Oracle Business Activity Monitoring, 50-29
 - public views, D-1
 - publish types, 17-2
 - sensor actions XSD schema file, D-5
 - variable sensors, 17-2
 - viewing metadata, 17-15
- sequence activity
 - capabilities, A-33
- sequence-next-val function
 - description, B-2
- sequential blocks
 - definition, 27-21
- sequential list of approvers
 - configuring, 27-35
- serial
 - definition, 27-35
 - workflow participant type, 27-35
- server connection, Oracle BAM, 51-2
- server.password parameter, 3-22
- server-setup-seed-deploy-test ant script, 3-23
- server.targets parameter, 3-22
- server.user parameter, 3-22
- Service Component Architecture
 - definition, 1-2
 - described, 1-3
- service components
 - adding, 2-6, 2-8
 - available types, 1-4
 - BPEL process, 1-4, 2-7, 4-1
 - business rules, 1-4, 2-7, 24-13
 - definition, 1-4
 - deleting, 2-8
 - editing, 2-9
 - human task, 1-4, 2-7, 27-1
 - introduction, 2-7, 2-11, 2-17
 - mediator, 1-4, 2-7, 18-1
 - metadata, 24-13
 - spring, 1-4, 2-7
 - types, 2-7
 - web service, 24-13
 - wiring, 2-20, 2-21
- Service Data Objects (SDO), 6-7
 - creating Enterprise JavaBeans integration with SOA composite applications, 35-8
 - definition, 1-2
 - converting from XML to SDO, 6-12
 - declaring SDO-based variables, 6-11
 - embedding with bpelx:exec, 13-8
 - entity variable support, 6-7
 - passing parameters between Enterprise JavaBeans and SOA composite applications, 35-1, 35-2
 - using in Enterprise JavaBeans Java interfaces
 - using in an Enterprise JavaBeans application, 35-3
 - using standalone SDO-based variables, 6-11
- service engines
 - definition, 1-7
 - described, 1-8
 - human workflow, 26-31
- Service Infrastructure
 - definition, 1-7
- service names
 - in adapters, 4-13
- Service-Oriented Architecture (SOA)
 - definition, 1-1
- services
 - adding, 2-10, 2-16
 - ADF-BC, 1-5, 34-11, A-42
 - AQ adapter, A-42
 - automatically exposing as a SOAP service, 2-10
 - database adapter, A-43
 - definition, 1-1, 1-5, 1-7, 2-12
 - deleting, 2-16
 - direct binding service, 34-12, 36-1, A-43
 - editing, 2-16
 - Enterprise JavaBeans (EJB) service, 34-11, A-43
 - file adapter, A-43
 - FTP adapter, A-43
 - HTTP binding, 34-5, A-43
 - JMS adapter, A-44
 - MQ adapter, A-44
 - Oracle Applications adapter, A-44
 - Oracle B2B, A-42
 - Oracle Business Activity Monitoring, A-42
 - overview, A-42
 - selecting a WSDL, 2-12
 - socket adapter, A-44
 - third party adapter, A-44
 - web service, A-44
 - wiring, 2-20
- servlet
 - adfdiRemote, 32-3
- setDomainEnv.cmd file, 3-4
- setDomainEnv.sh file, 3-4
- setting folder permissions, 52-11
- setting up, 31-32
- signal activity
 - capabilities, A-34
 - in master and detail processes, 15-7
- simple types
 - supported as message parts, 2-14
- single approver
 - configuring, 27-23
 - definition, 27-23
 - workflow participant type, 27-23
- Skip Condition tab
 - bypassing execution of activities, 10-13
 - in activities, A-6
 - only available in BPEL 1.1 projects, A-6
- SMS activity
 - capabilities, A-35
 - notifications support, 16-10
- SOA Composer

- accessing, 45-2
- committing changes at runtime, 45-5
- definition, 45-1
- detecting conflicts among concurrent users, 45-6
- editing domain value maps at runtime, 45-4
- saving domain value maps at runtime, 45-5
- SOADesigner role required to access metadata, 45-3
- viewing domain value maps at runtime, 45-3
- SOA composite applications
 - activating, 2-27
 - creating, 2-1
 - customizing, 43-1
 - deploying a single composite, 40-14
 - deploying an existing archive, 40-39
 - deploying multiple composites, 40-27
 - deploying shared metadata across composites, 40-29
 - deployment, 2-28
 - interacting with Enterprise JavaBeans, 35-1
 - invoking other composites, 2-25
 - invoking the default revision, 2-19
 - restrictions on application names, 2-2
 - retiring, 2-27
 - setting as the default revision, 2-28
 - shutting down, 2-27
 - starting up, 2-27
 - testing, 2-25
 - undeploying, 2-28
- SOA Composite Editor
 - layout, 2-4
- SOA Governance
 - Oracle Enterprise Repository, A-55
- SOA Infrastructure
 - properties, K-5
 - System MBean Browser properties, K-7
- SOA XPath extension functions, B-1
- SOA-MDS connections
 - opening the composite.xml file, 2-6
- soa.only.deployment parameter, 3-22
- SOAP
 - definition, 1-3
 - reading and encoding SOAP attachment content, 42-7
 - security in SOAP web services, 31-5
 - support in workflow services, 31-2
 - with attachments, 42-3
- SOAP headers, 6-53
 - receiving in BPEL, 6-53
 - sending in BPEL, 6-54
- SOAP services
 - performance issues, 13-1
 - using Java code, 13-1
- SOAP-encoded arrays, 6-46
 - in BPEL 2.0, 6-47
- soa.server.oracle.home parameter, 3-22
- socket adapter
 - capabilities, A-44
 - definition, 34-10
- Source window
 - location of in Oracle JDeveloper, 4-5
- sources
 - message, 53-1
- Sources tab
 - in activities, A-6
- specifying operation or event subscription properties, 18-33
- spring
 - contents of componentType file, 49-13
 - contents of spring context file, 49-7
 - creating a spring service component in Oracle JDeveloper, 49-5
 - EXM files, 49-29
 - in Fusion Order Demo, 3-7, 49-21
 - integration
 - of Java and WSDL-based components in the same composite, 49-2
 - introduction, 49-1
 - JAXB and OXM support, 49-28
 - service component, 2-7
 - using callbacks, 49-4
- square-root function
 - description, B-45
- SSL
 - configuring when creating an application server connection, 40-15
- stages
 - definition, 27-20
- standard faults
 - BPEL 1.1, 11-3
 - BPEL 2.0, 11-3
 - definition, 11-3
- Store Front module
 - deploying, 3-19
 - fod.application.issuaenabled property, 3-17
 - placing orders, 3-23
 - StoreFrontService project, 3-2
- StoreFrontService project, 3-2
- StoreFrontUI project, 3-2
- streamIncomingAttachments
 - property, K-3
- streamIncomingAttachments property, 42-6
- streaming
 - attachments, 42-3
 - Oracle B2B, 42-12
 - properties for streaming attachments, 42-6
 - sending attachment streams, 42-7
 - with the file and FTP adapters, 42-11
- streamOutgoingAttachments
 - property, K-3
- streamOutgoingAttachments property, 42-6
- string functions in calculations, 52-5
- strings
 - concatenating, 6-18
 - converting to an XML element, 6-51
- Structure window
 - location of in Oracle JDeveloper, 4-5
- subtract-dayTimeDuration-from-dateTime function
 - description, B-6
- switch activity

- capabilities, A-35
- in conditional branching logic, 10-2
- replaced by the if activity in BPEL 2.0, 7-3, A-4
- synchronization
 - of activity execution, 9-5
- synchronous callbacks, 7-1
 - operational concepts, 7-2
 - SyncMaxWaitTime property, 7-6
- synchronous interactions
 - BPEL process as the client, 5-3
 - BPEL process as the service, 5-3
 - definition, 5-2
 - returning faults, 11-28
- synchronous processes
 - calling a one-way mediator, 7-7
 - setting timeouts, 14-15
- synchronous receiving, 60-7, 61-9, 62-10
- synchronous requests
 - not timing out, 7-6
- synchronous services
 - callbacks with the partner link and invoke activity, 7-1
 - calling, 7-2
 - invoke activities, 7-5
 - ports, 7-1
- SyncMaxWaitTime property
 - in synchronous callbacks, 7-6
 - synchronous requests not timing out, 7-6
- System MBean Browser
 - properties, K-7

T

- Targets tab
 - in activities, A-6
- task action time limits
 - specifying, 27-30, 27-35, 27-37
- task admin
 - definition, 26-7
- task assignments
 - dynamic, 26-6
 - restricting, 27-73
 - rule-based, 26-6
 - static, 26-6
- task category
 - specifying, 27-11
- task conditions
 - abruptly completing a condition, 27-42
- task deadlines
 - definition, 26-7
- task display form
 - autogenerated, 28-9
 - creating, 28-9, 28-26, 28-28
 - creating a task form with the Custom Task Form Wizard, 28-12
 - definition, 27-3, 28-1
 - deploying, 28-38
 - displaying, 28-47
 - generating content for facets, 28-13
 - registering the library JAR file for custom page templates, 28-11
- .task file
 - associating with a BPEL process, 27-2, 27-78
 - definition, 27-2, 27-5
- task flow
 - ADF
 - task display form for human tasks, 28-4
 - deploying, 40-21
- task history
 - specifying in human task activities, 27-85
- task initiator
 - definition, 26-7
 - specifying, 27-81
- task instance attributes, 31-23
- task list portlets
 - assignment filter constraints, 33-20
 - configuring EJB identity propagation, 33-5
 - configuring the identity store, 33-5
 - connecting the task list producer to the remote SOA server, 33-3
 - creating a portlet consumer application for embedding the task list portlet, 33-9
 - defining the foreign JNDI provider, 33-3
 - deploying the task list producer application on a portlet server, 33-2
 - deployment prerequisites, 33-2
 - example of file containing all column constraints, 33-21
 - introduction, 33-1
 - passing worklist portlet parameters, 33-16
 - securing the task list portlet producer application, 33-6
 - specifying the inbound security policy, 33-7
- task metadata service
 - definition, 26-28
 - Enterprise JavaBeans, SOAP, and Java support, 31-2
 - supported task operations, 31-13
 - WSDL file location, 31-2
- task notification
 - editing notification messages, 27-64
 - making email actionable, 31-32
 - notifying recipients of changes to task status, 27-62
 - overview, 27-60
 - reminders, 31-34
 - securing notifications, 31-34
 - setting up reminders, 27-65
 - task attachments with email notifications, 31-34
- task outcome
 - specifying, 27-9
- task outcomes
 - restrictions on specifying custom names, 27-10
- task owner
 - definition, 26-7
 - specifying by browsing the user directory, 27-12
 - specifying in human task activities, 27-85
 - specifying through XPath expressions, 27-16
- task parameters
 - specifying, 27-81

- task participants
 - allowing all participants to invite other participants, 27-42
 - assigning task participants by name or expression, 27-26, 27-53
 - bypassing, 27-31, 27-35, 27-38
 - dynamically assigning with the assignment service, 31-39
 - inviting additional task participants, 27-31, 27-35, 27-37
 - sharing attachments and comments, 27-34
 - specifying, 27-20
- task payload data structure
 - specifying, 27-17
- task priority
 - specifying, 27-11, 27-81
- task query service
 - definition, 26-28
 - Enterprise JavaBeans, SOAP, and Java support, 31-2
 - supported task operations, 31-9
 - WSDL file location, 31-2
- task reminders
 - setting up, 27-65
- task report service
 - Enterprise JavaBeans, SOAP, and Java support, 31-2
 - supported task operations, 31-16
 - WSDL file location, 31-3
- task reviewer
 - definition, 26-7
- task routing service
 - definition, 26-27
- task service
 - definition, 26-27
 - Enterprise JavaBeans, SOAP, and Java support, 31-2
 - supported task operations, 31-6
 - WSDL file location, 31-2
- task stages
 - definition, 26-9
- task title
 - specifying, 27-80
- tasks
 - escalating, renewing, or ending a task, 27-55
 - notifications and reminders, 31-27
- TCP tunneling
 - setting up a TCP listener for asynchronous services, 8-14
 - setting up a TCP listener for synchronous services, 8-13
- terminate activity
 - capabilities, A-36
 - definition, 11-43
 - fault handling, 11-43
 - replaced by the exit activity in BPEL 2.0, A-4
- test suites
 - components, 41-3
 - creating, 41-5
 - definition, 41-1
 - limitations on multibyte character names, 41-5
- third party adapter
 - capabilities, A-44
 - definition, 34-10
- thread.sleep()
 - using in a Java embedding activity, 13-8
- throw activity
 - capabilities, A-37
 - throwing internal faults, 11-25
- time
 - assigning with a function, 6-20
- time dimensions, 52-16
- time duration format, 14-2
- time stamp field, 52-6
- time zones, changing, 29-75
- Timeout tab
 - in activities, 14-7, A-6
 - only available in BPEL 1.1 projects, A-6
 - setting for request-response operations in receive activities, 14-7
- timeout values
 - specifying, 7-6
- timeouts
 - event added to the audit trail during a timeout, 14-10
 - increasing the JTA transaction timeout value, 42-12
 - of BPEL processes, 14-1
 - recoverable timeout activities during a server restart, 14-10
 - setting for request-response operations in receive activities, 14-7, 14-10
 - setting relative from when the activity is invoked, 14-7
 - settings as an absolute date time, 14-8
 - settings computed dynamically with an XPath expression, 14-9
 - SyncMaxWaitTime property, 7-6
 - using pick activities, 14-1
 - using the wait activity, 14-12
- timezone-from-dateTime function
 - description, B-7
- title
 - specifying in a human task, 27-8
- top-down design approach, 1-8
- trackable fields
 - composite sensors, 47-1
- transaction property
 - description, C-2
 - setting, 12-1, 12-2, 12-3
- transaction semantics
 - in BPEL processes, 12-1
- transaction timeouts
 - increasing the JTA transaction timeout value, 42-12
- transform activity
 - capabilities, A-37
 - creating, 37-7
- transformations
 - adding XSLT constructs, 37-25

- auto mapping, 37-32
- auto mapping with confirmation, 37-33
- chaining functions, 37-20
- correcting memory errors, 37-49
- creating, 37-7
- creating a map file from imported schemas, 37-9
- creating a new map file, 37-7
- creating an XSL map from an XSL style sheet, 37-7
- customizing sample XML generation, 37-50
- dictionaries, 37-36
- editing functions, 37-20
- editing XPath expressions, 37-24
- error when mapping duplicate elements, 37-7
- functions, 37-19
- functions prefixed with xp20 or orcl, 37-19
- generating optional elements, 37-50
- generating reports, 37-48
- ignoring elements, 37-41
- linking source target nodes, 37-17
- map parameter and variable creation, 37-37
- named templates in functions, 37-21
- repeating elements, 37-28
- replacing schemas, 37-41
- rules, 37-6
- searching source and target nodes, 37-39
- setting constant values, 37-18
- setting the maximum depth, 37-50
- setting the number of repeating elements, 37-50
- testing the map file, 37-45
- using arrays, 37-28
- using the XSLT Mapper, 37-16
- using XQuery and XSLT, 6-5
- viewing unmapped target nodes, 37-35
- xsl choose conditional processing, 37-28
- xsl if conditional processing, 37-26
- translateFromNative function
 - description, B-45
- translateToNative function
 - description, B-46
- troubleshooting
 - deployment, 40-62
- tuning
 - general recommendations, 42-12
- two-layer business process management
 - definition, 48-1
 - dynamic routing decision table, 48-6
 - phase activity, 48-3
 - use case, 48-7

U

- UDDI *See* Oracle Service Registry
- uddiCacheLifetime
 - property, K-9
- undeployment
 - SOA composite applications, 2-28
- Unicode support, 2-2
- upper-case function
 - description, B-14

- user directory
 - selecting notification recipients by browsing the directory, 16-13
- user metadata service
 - definition, 26-28
 - Enterprise JavaBeans, SOAP, and Java support, 31-2
 - supported task operations, 31-14
 - WSDL file location, 31-3
- user notification activity
 - allowing the end user to select the notification channels, 16-14
 - capabilities, A-38
- user notifications
 - definition, 16-14
- using domain value maps, 44-8
- using domain value maps a transformation, 44-9
- using error handling, 21-10
- using lookupValue functions, 44-11
- using Oracle Mediator error handling, 21-10

V

- validate activity
 - capabilities, A-39
- validate syntax (XSD) property
 - specifying operation or event subscription properties, 18-33
- validateXML property
 - description, C-3
- validation
 - of XML data with bpelx:validate, 6-35
 - when loading a process diagram, A-55
- variable sensors
 - definition, 17-2
- variables
 - complex type, 6-15
 - copying data between, 6-14
 - element variables in message exchange activities in BPEL 2.0, 6-36
 - initializing variables inline in BPEL 2.0, 8-5
 - initializing with an inline from-spec in BPEL 2.0 projects, 6-15
 - initializing with expression constants, 6-13
 - initializing with literal XML, 6-13
- ViewController project, 32-2
- voice activity
 - capabilities, A-40
 - notifications support, 16-12
- voice mail
 - dynamically setting telephone numbers, 16-12
 - notifications support, 16-12

W

- wait activity
 - capabilities, A-40
 - creating, 14-12
 - definition, 14-12
 - setting an expiration time, 14-12

- web services
 - adding a WSDL file, 2-12
 - binding component, 2-11, 2-17
 - capabilities, A-44
 - connecting with SOAP over HTTP, 1-5
 - DataObjectDefinition, 56-3
 - DataObjectOperations, 56-2
 - definition, 34-2
 - ICommand, 56-4
 - ManualRuleFire, 56-4
 - service component, 24-13
 - WS-Atomic transactions support, 34-2
 - WSDL files, 24-13
- WebLogic Fusion Order Demo application
 - B2BX12OrderGateway project, 3-7
 - bin project, 3-7
 - composite.xml file, 3-7
 - CreditCardAuthorization project, 3-7
 - deploying, 3-20
 - ExternalLegacyPartnerSupplier project, 3-7
 - OrderApprovalHumanTask project, 3-7
 - OrderBookingComposite project, 3-7
 - OrderSDOComposite project, 3-7
 - overview, 3-6
 - PartnerSupplierComposite project, 3-7
 - processing described, 3-8
 - projects in, 3-7
 - setting up, 3-3
 - viewing in Oracle JDeveloper, 3-6
- web.xml file, 32-3
- wfDynamicGroupAssign function
 - description, B-61
- wfDynamicUserAssign function
 - description, B-62
- while activity
 - capabilities, A-41
 - in conditional branching logic, 10-8
- wires
 - adding, 2-20, 2-23
 - definition, 1-6
 - deleting, 2-23
 - using, 2-20
 - wiring a service component and reference, 2-21
- WLST utility
 - creating a configuration plan, 40-14
 - deployment with, 40-41
- WordML style sheets
 - using for attachments, 27-54
- workflow context
 - creating on behalf of a user, 31-5
- workflow functions
 - overview, 31-1
- workflow service clients, 30-3
 - interface, 30-5
- workflow services
 - abruptly completing a condition, 27-42
 - actionable emails, 31-32
 - allowing all participants to invite other participants, 27-42
 - assigning task participants by name or expression, 27-26, 27-53
 - assignment service configuration, 31-36
 - associating the human task activity with a BPEL process, 27-78
 - associating the human task definition with a BPEL process, 27-2
 - bypassing task participants, 27-31, 27-35, 27-38
 - changing character set encoding, 27-65
 - customizing notification headers, 27-67
 - editing notification messages, 27-64
 - Enterprise JavaBeans support, 31-2
 - escalate after policy, 27-58
 - escalating, renewing, or ending a task, 27-55
 - escalation and expiration policy overview, 27-56
 - escalation rules, 27-59
 - expire after policy, 27-57
 - functions, B-58
 - clearTaskAssignees, B-58
 - createWordMLDocument, B-58
 - getNotificationProperty, B-58
 - getNumberOfTaskApprovals, B-59
 - getPreviousTaskApprover, B-59
 - getTaskAttachmentByIndex, B-59
 - getTaskAttachmentByName, B-60
 - getTaskAttachmentContents, B-60
 - getTaskAttachmentsCount, B-60
 - getTaskResourceBindingString, B-61
 - wfDynamicGroupAssign, B-61
 - wfDynamicUserAssign, B-62
 - FYI assignee task participant, 27-38
 - group voting details, 27-33
 - identification key, 27-85
 - identity service, 26-28
 - including the task history of other tasks, 27-85
 - inviting additional task participants, 27-31, 27-35, 27-37
 - Java support, 31-2
 - making email messages actionable, 27-66
 - multilingual settings, 27-54, 31-31
 - never expire policy, 27-57
 - notification contents, 31-28
 - notification preferences, 27-60
 - notification service, 26-28, 31-30
 - notifications, 31-27
 - notifying recipients of changes to task status, 27-62
 - overview, 31-1
 - parallel task participant, 27-31
 - renew after policy, 27-58
 - routing slip
 - definition, 27-30
 - runtime config service, 26-28
 - scope name and global task variable name, 27-84
 - securing notifications, 27-65, 31-34
 - security model, 31-4, 31-5
 - sending email notifications to groups and application roles, 27-66
 - sending task attachments with email notifications, 27-66
 - serial task participant, 27-35

- setting up reminders, 27-65
- sharing attachments and comments with task participants, 27-34
- showing the Oracle BPM Worklist URL in notifications, 27-65
- single approver task participant, 27-23
- SOAP support, 31-2
- specifying a task initiator and task priority, 27-81
- specifying a task title, 27-80
- specifying callback classes, 27-74
- specifying task parameters, 27-81
- support for identity service, 31-11
- task attachments with email notifications, 31-34
- task category, 27-11
- task display form, 27-3, 28-1
- .task file
 - definition, 27-2, 27-5
- task metadata service, 26-28
- task notifications, 31-27
- task outcomes, 27-9
- task owner, 27-85
- task owner specification through the user directory, 27-12
- task owner specification through XPath expressions, 27-16
- task participants, 27-20
- task payload data structure, 27-17
- task priority, 27-11
- task query service, 26-28
- task routing and customization in BPEL callbacks, 27-77
- task routing service, 26-27
- task service, 26-27
- task title, 27-8
- time limits for acting on tasks, 27-30, 27-35, 27-37
- user metadata service, 26-28
- viewing BPEL callbacks, 27-87
- WordML style sheets in attachments, 27-44
- worklist
 - acting on tasks, 29-28
 - acting on tasks that require a digital signature, 29-36
 - administration functions, 29-46
 - approving tasks, 29-40
 - assignment rules for tasks with multiple assignees, 29-45
 - creating a subtask, 29-21
 - creating a ToDo list, 29-20
 - creating and customizing worklist views, 29-15
 - creating group rules, 29-44
 - creating user rules, 29-43
 - customizing the task status chart, 29-19
 - definition, 29-1
 - filtering tasks, 29-8
 - logging in, 29-3
 - managing messaging channels, 29-56
 - managing messaging filters, 29-58
 - managing rules, 29-46
 - mapping mapped attributes, 29-62
 - messaging filter rules, 29-55
 - reports, 29-66, 29-67
 - rule actions, 29-56
 - setting a vacation period, 29-41
 - setting rules, 29-42
 - specifying notification settings, 29-54
 - system actions, 29-25
 - Task Details page, acting on tasks, 29-22
 - task history, 29-26
 - Task Listing page contents, 29-6
 - Task Listing page, customizing, 29-8
 - using mapped attributes, 29-61
- worklist clients
 - building for workflow services, 30-1
 - class paths for clients using remote Enterprise JavaBeans, 30-6
 - class paths for clients using SOAP, 30-6
 - customizing, 30-1
 - packages and classes for, 30-2
- writeBinaryToFile function
 - description, B-40
- WS-Addressing
 - sending correlation IDs, 8-8
 - using in an asynchronous service, 8-12
- WS-Atomic transactions
 - composite.xml file syntax, 34-4
 - enabling participation of BPEL processes, 34-4
 - not supported when optimization is enabled, 34-4
 - support in SOA composite applications, 34-2
- wsclient.jar file, 32-2
- WSDL files
 - adding for a web service, 2-12
 - definition, 1-3
 - differences between document-literal styles and RPC styles, 6-1, 6-52
 - editing in Source View is not supported, 2-15
 - integration of Java and WSDL-based components in the same SOA composite application, 49-2
 - invoking the default revision, 2-19
 - limitation on mixed message types in a WSDL file, 2-19
 - location for evidence store service, 31-3
 - location for identity service, 31-2
 - location for runtime config service, 31-3
 - location for task metadata service, 31-2
 - location for task query service, 31-2
 - location for task report service, 31-3
 - location for task service, 31-2
 - location for user metadata service, 31-3
 - modifying to generate a fault, 11-24
 - references, 2-19
 - selecting, 2-12
 - service component metadata, 24-13
 - specifying when creating a partner link, 4-8
 - updating message schemas, 2-15
 - using an existing WSDL file, 2-13
 - viewing message schemas, 2-15
 - with multiple parts, 2-14
 - WSDL namespaces must be unique, 2-15

X

XML assert

- overview, 41-2

XML data in BPEL, 6-2

XML data manipulation

- bpelx:append extension, 6-23
- bpelx:copyList extension, 6-31
- bpelx:insertAfter extension, 6-26
- bpelx:insertBefore extension, 6-24
- bpelx:remove extension, 6-27
- bpelx:rename extension, 6-29
- bpelx:validate extension, 6-35

XML documents

- manipulating, 6-2, 6-5
- overview, 6-2, 6-5

XML facades

- definition, 13-4
- Java embedding, 13-4

XML schema files

- error handling, 21-11
- fault-bindings.xml, 21-15
- fault-policies.xml, 21-11

XML schemas

- message types and variable types, 6-1

XPath Building Assistant

- using, B-67
- using in the XSLT Mapper, 37-24, B-70

XPath expressions

- assigning numeric values, 6-17
- boolean expressions in switch activities, 10-4
- dynamically creating another XPath expression, 6-48
- dynamically setting email addresses and telephone numbers, 16-13
- editing in transformations, 37-24
- examples, 6-4
- fetching a data sequence element, 6-48
- in conditional branching logic, 10-1
- specifying a task owner, 27-16

XPath extension functions

- creating user-defined functions, B-74
- dvm
 - lookupValue function, 44-8
 - lookupValue1M function, 44-9

XPath functions

- in transformations, 37-19
- indexing methods, 6-46
- mathematical calculations, 6-17

XPath queries

- copying data, 6-15
- examples, 6-4

XQuery, 6-2, 6-5

xref

- lookupXRef function, 46-15
 - exception reasons, 46-16
 - parameters, 46-15
- lookupXRef1M function
 - exception reasons, 46-17, 46-18
 - parameters, 46-16, 46-17
- markForDelete function, 46-19

- exception reasons, 46-20

- parameters, 46-20

populateXRefRow function

- modes, 46-11

- parameters, 46-11

populateXRefRow1M function, 46-12

- modes, 46-13

- parameters, 46-12

xsl choose

- conditional processing, 37-28

xsl if

- conditional processing, 37-26

XSL map

- creating from an XSL style sheet, 37-7

XSL style sheet

- creating an XSL map, 37-7

XSL transformations

- definition, 1-3

XSLT, 6-2, 6-5

XSLT constructs

- adding in transformations, 37-25

XSLT Mapper

- adding XSLT constructs, 37-25
- auto mapping, 37-32
- auto mapping with confirmation, 37-33
- chaining functions, 37-20
- correcting memory errors when generating reports, 37-49
- creating a map file, 37-1
- creating a map file from imported schemas, 37-9
- creating a new map file, 37-7
- creating a transform activity, 37-7
- creating an XSL map from an XSL style sheet, 37-7
- customizing sample XML generation for transformations, 37-50
- dictionaries, 37-36
- editing functions, 37-20
- editing XPath expressions, 37-24
- error when mapping duplicate elements, 37-7
- functions, 37-19
- functions prefixed with xp20 or orcl, 37-19
- generating optional elements, 37-50
- generating reports, 37-48
- ignoring elements, 37-41
- layout in Oracle JDeveloper, 37-1
- linking source and target nodes, 37-17
- map parameter and variable creation, 37-37
- named templates in functions, 37-21
- repeating elements, 37-28
- replacing schemas, 37-41
- rules, 37-6
- searching source and target nodes, 37-39
- setting constant values, 37-18
- setting the maximum depth, 37-50
- setting the number of repeating elements, 37-50
- testing the map file, 37-45
- using, 19-56, 37-16
- using arrays, 37-28
- viewing unmapped target nodes, 37-35

xsl choose conditional processing, 37-28
xsl if conditional processing, 37-26

Y

year-from-dateTime function
description, B-7