Oracle® Fusion Middleware

Developer's Guide for Oracle SOA Suite

11*g* Release 1 (11.1.1.5.0)

E10224-09

June 2011

ORACLE®

Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite, 11*g* Release 1 (11.1.1.5.0)

E10224-09

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

Contents

3 Introduction to the SOA Sample Application

Part II Using the BPEL Process Service Component

4 Getting Started with Oracle BPEL Process Manager

5 Introduction to Interaction Patterns in a BPEL Process

6 Manipulating XML Data in a BPEL Process

7 Invoking a Synchronous Web Service from a BPEL Process

8 Invoking an Asynchronous Web Service from a BPEL Process

9 Using Parallel Flow in a BPEL Process

10 Using Conditional Branching in a BPEL Process

11 Using Fault Handling in a BPEL Process

15 Coordinating Master and Detail Processes

16 Using the Notification Service

19 Creating Oracle Mediator Routing Rules

20 Working with Multiple Part Messages in Oracle Mediator

21 Using Oracle Mediator Error Handling

22 Resequencing in Oracle Mediator

25 Using Declarative Components and Task Flows

Part V Using the Human Workflow Service Component

26 Getting Started with Human Workflow

27 Designing Human Tasks

28 Designing Task Forms for Human Tasks

29 Using Oracle BPM Worklist

30 Building a Custom Worklist Client

31 Introduction to Human Workflow Services

32 Integrating Microsoft Excel with a Human Task

33 Configuring Task List Portlets

Part VI Using Binding Components

34 Getting Started with Binding Components

35 Integrating Enterprise JavaBeans with SOA Composite Applications

36 Using the Direct Binding Invocation API

Part VII Sharing Functionality Across Service Components

37 Creating Transformations with the XSLT Mapper

38 Using Business Events and the Event Delivery Network

41 Automating Testing of SOA Composite Applications

43 Customizing SOA Composite Applications

44 Working with Domain Value Maps

45 Using Oracle SOA Composer with Domain Value Maps

46 Working with Cross References

47 Defining Composite Sensors

48 Using Two-Layer Business Process Management (BPM)

49 Integrating the Spring Framework in SOA Composite Applications

Part X Using Oracle Business Activity Monitoring

50 Integrating Oracle BAM with SOA Composite Applications

51 Using Oracle BAM Data Control

52 Defining and Managing Oracle BAM Data Objects

53 Creating Oracle BAM Enterprise Message Sources

54 Using Oracle Data Integrator With Oracle BAM

55 Creating External Data Sources

56 Using Oracle BAM Web Services

57 Creating Oracle BAM Alerts

58 Using ICommand

Part XI Using Oracle User Messaging Service

59 Oracle User Messaging Service

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

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

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

63 Parlay X Web Services Multimedia Messaging API

B XPath Extension Functions

C Deployment Descriptor Properties

D Understanding Sensor Public Views and the Sensor Actions XSD

E Oracle BAM Web Services Operations

F Oracle BAM Alert Rule Options

G Oracle BAM ICommand Operations and File Formats

H Normalized Message Properties

I Interfaces Implemented By Rules Dictionary Editor Task Flow

J Oracle User Messaging Service Applications

K Oracle SOA Suite Properties Road Map

Index

Preface

This manual describes how to use Oracle SOA Suite.

This preface contains the following topics:

Audience

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

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

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.
italic	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:

1

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:

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-opencsa.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.

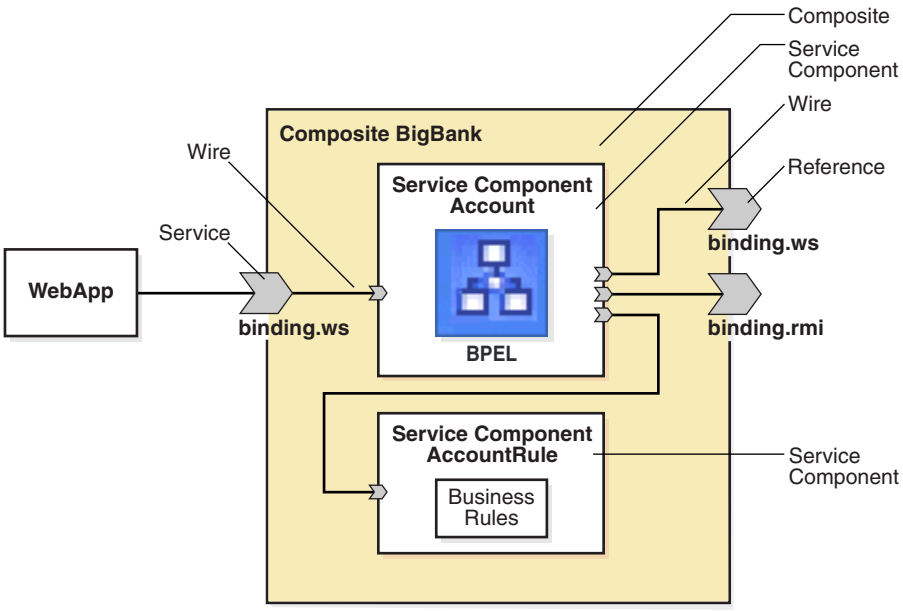
- Easy Maintenance and Debugging

  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:

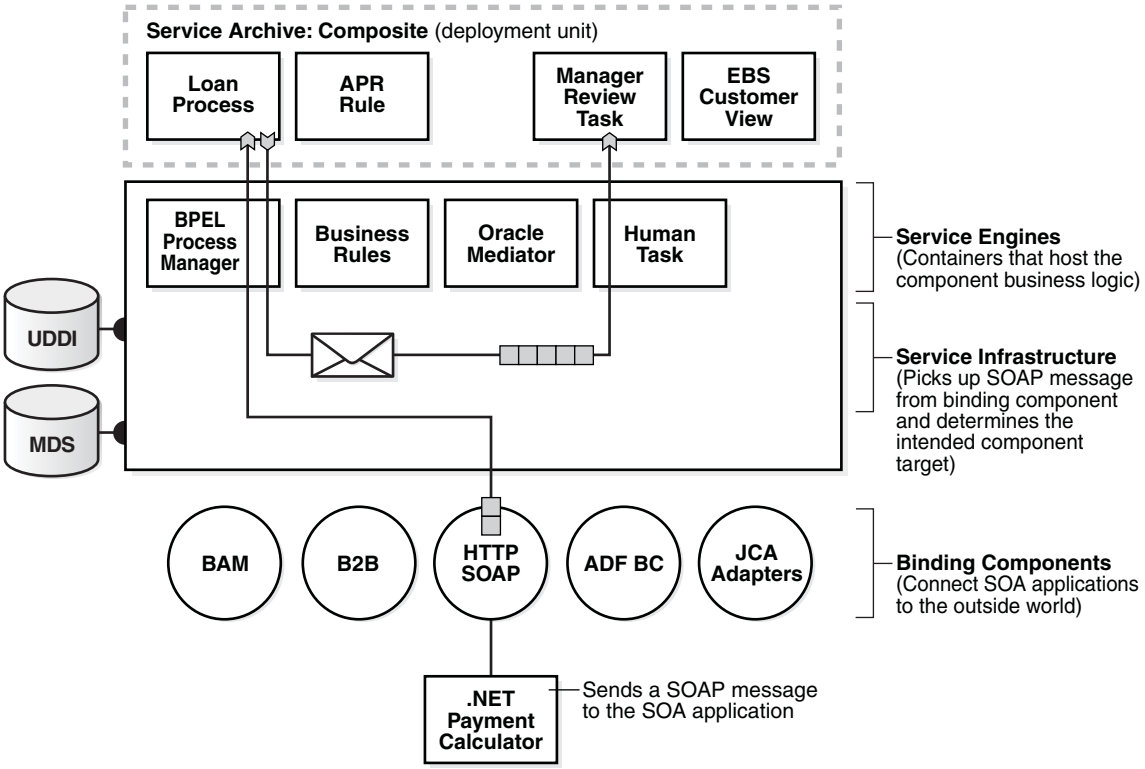
- Services to service components
- Service components to other service components
- Service components to references

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: ■ *Service* binding components provide an entry point to the SOA composite application. ■ *Reference* binding components enable messages to be sent from the SOA composite application to external services.	The SOAP binding component *service*: ■ 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 *reference* 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.	*Oracle Fusion Middleware Getting Started with Oracle SOA Suite*
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 11*g* 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*.

2

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:

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

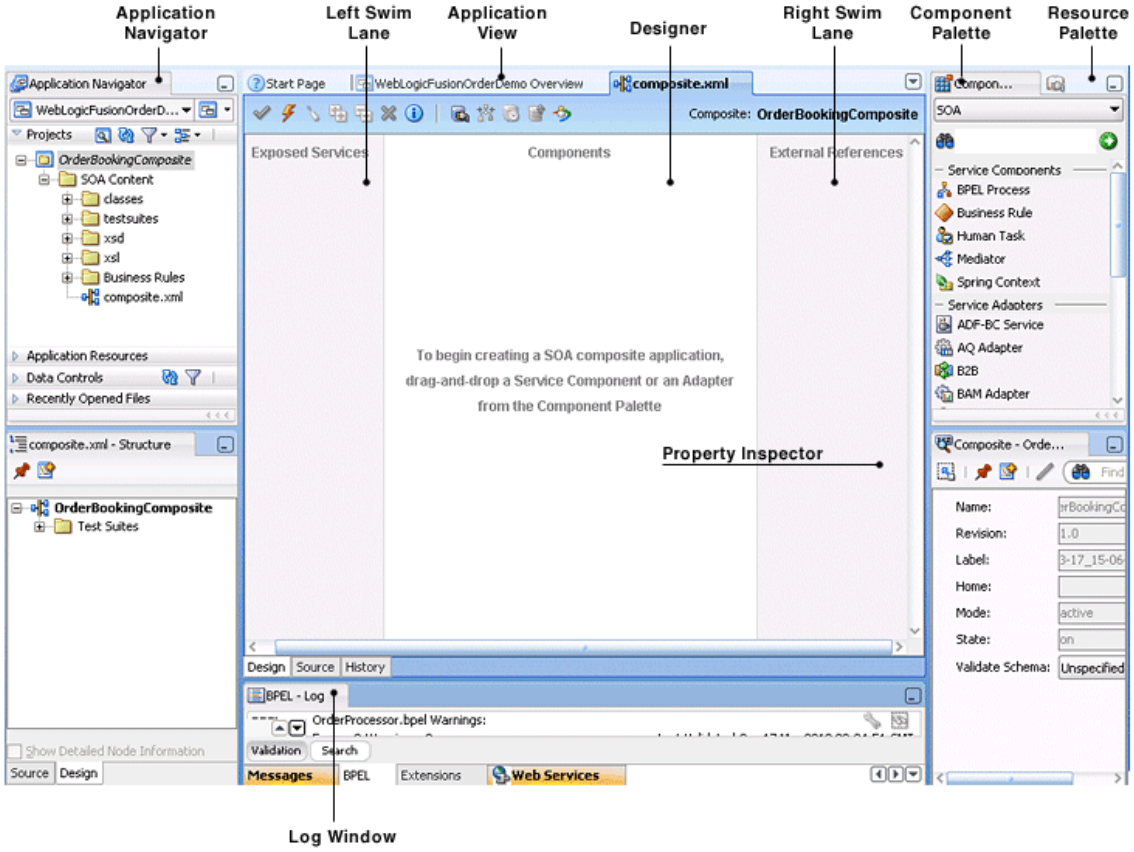


Table 2–3 describes the SOA Composite Editor.

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 (*rules_name*.decs). Additional business rules files display under the **Oracle** > **rules** subfolder (*rules_name*.rules). ▪ The Oracle Mediator service component file (*mediator_name*.mplan). ▪ The BPEL process service component files (*process_name*.bpel and *process_name*.wsdl). ▪ The human task service component file (*task_name*.task). ▪ The spring service component file (*spring*.xml). ▪ 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.

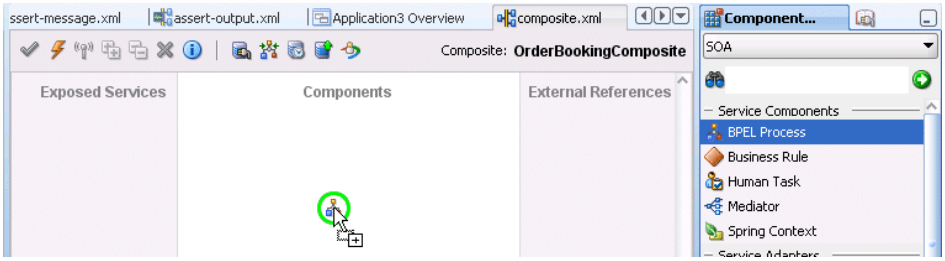
2.2.1 How to Add a Service Component

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.

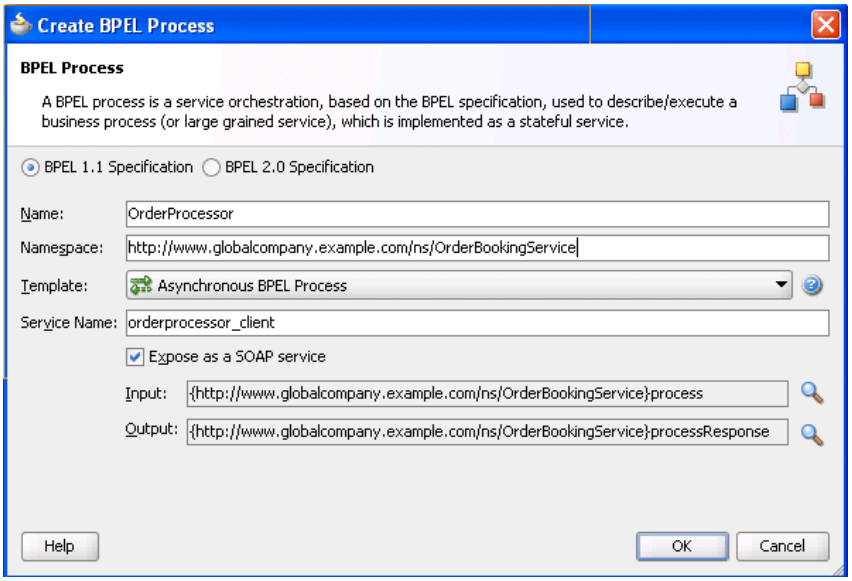
Table 2–4 Starting Service Component Editors

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.

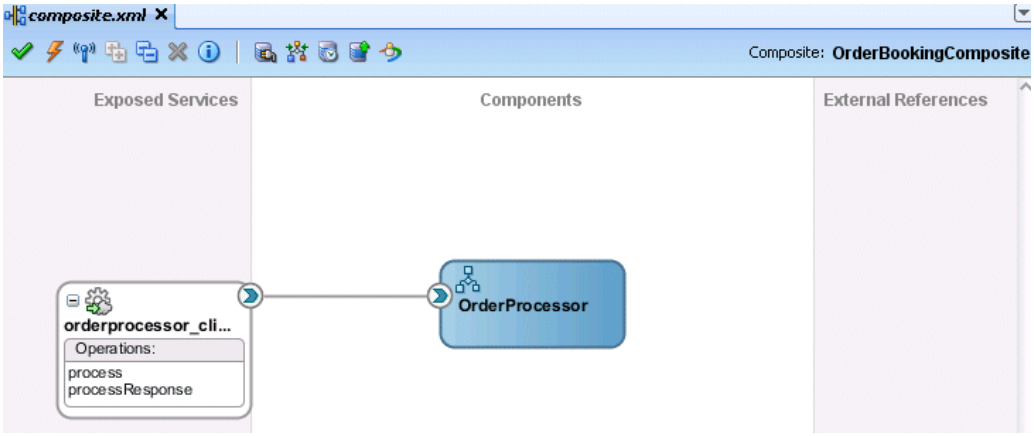
Figure 2–3 Create BPEL Process Dialog



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.

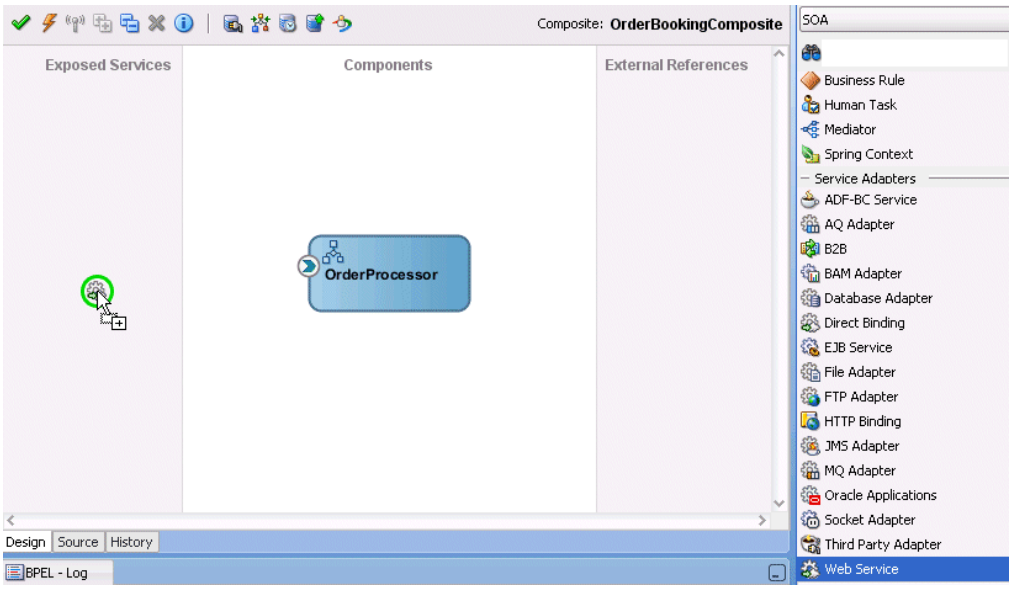
You can use the Component Palette from the SOA Composite Editor to drag and drop service binding components to the composite.

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.

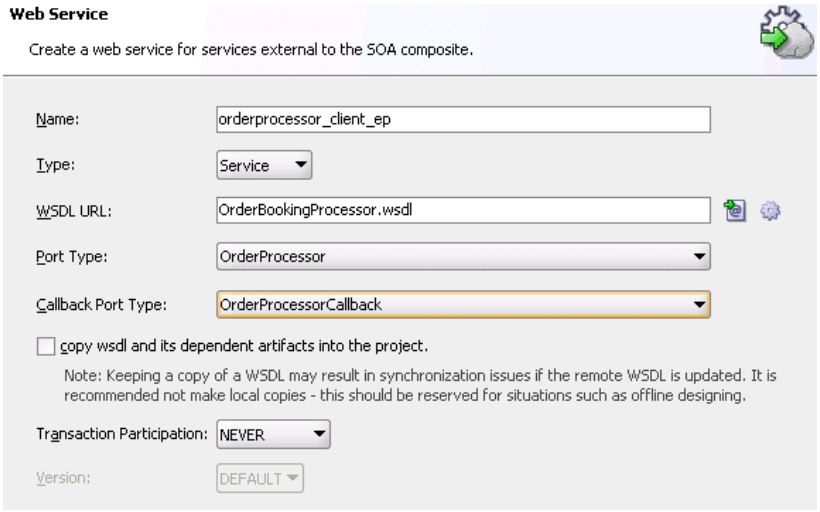
Table 2–6 Service Editors

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.

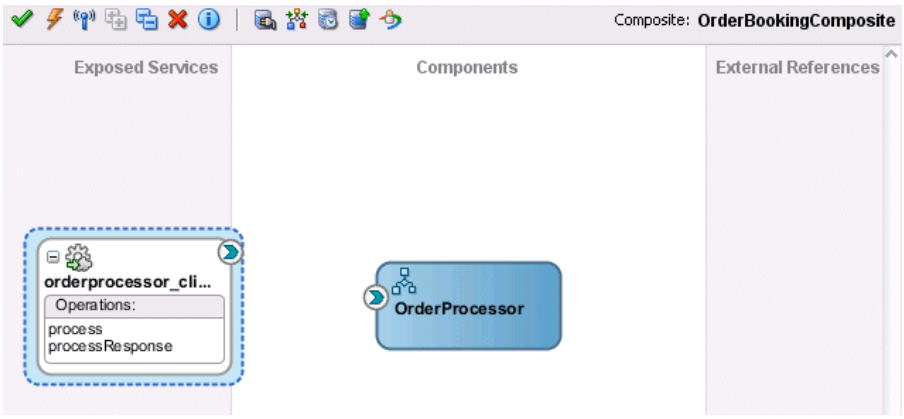
Figure 2–6 Create Web Service Dialog



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.

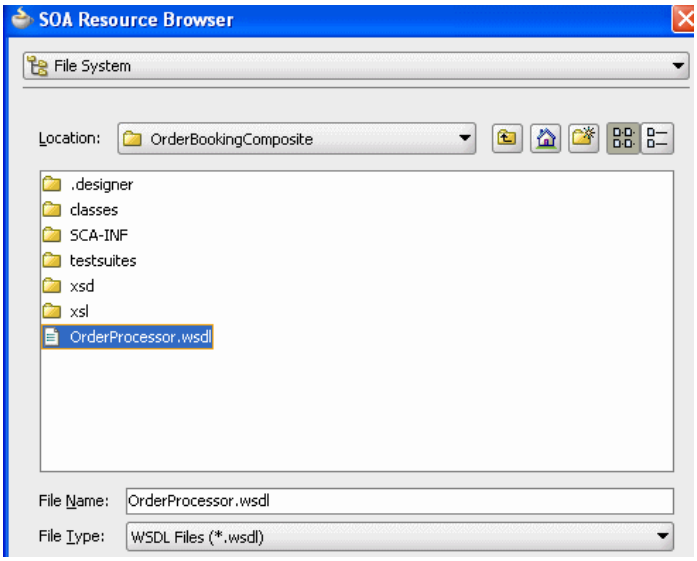
   This invokes the Create Web Service dialog shown in Figure 2–6.
3. Enter the details shown in Table 2–7:

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.

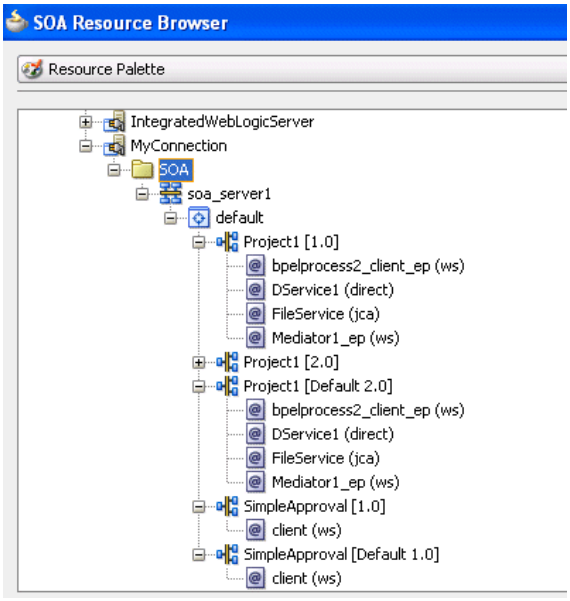
4. Select the WSDL file for the service. There are three methods for selection:
   a. To the right of the **WSDL URL** field, click the first icon and select an existing WSDL file from the local file system (for this example, **OrderProcessor.wsdl** is selected). Note that **File System** in the list 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.

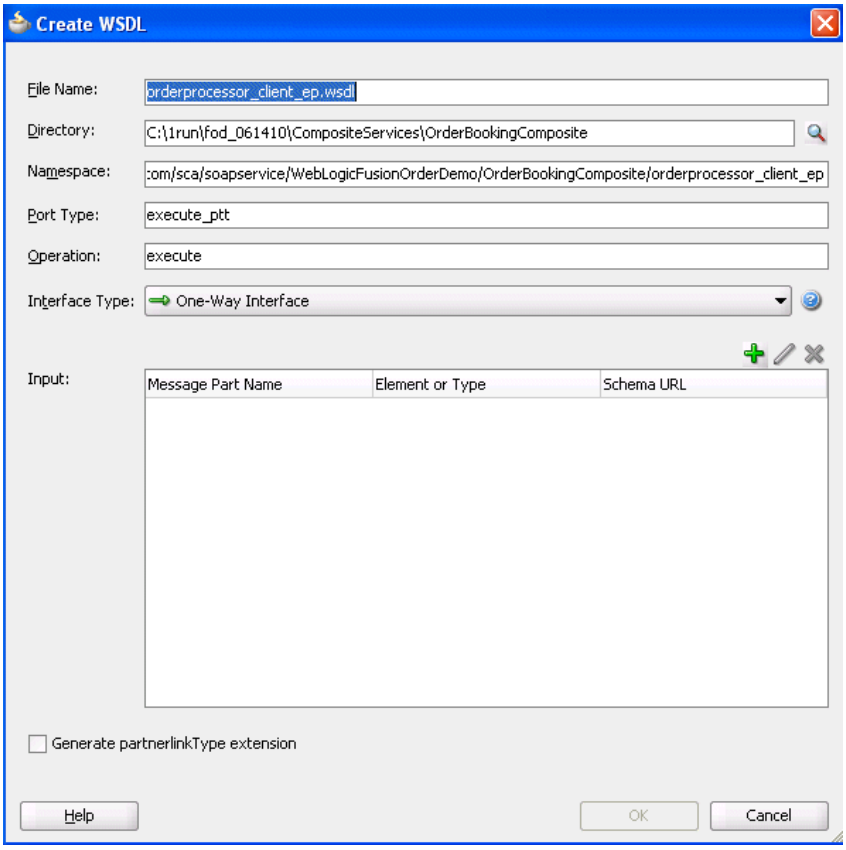
Figure 2–9 Use of Existing WSDL files from Other Applications



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



5. 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**.

6. Click **OK** to return to the Create Web Service dialog.

7. 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.

8. Click **OK**.

9. 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.

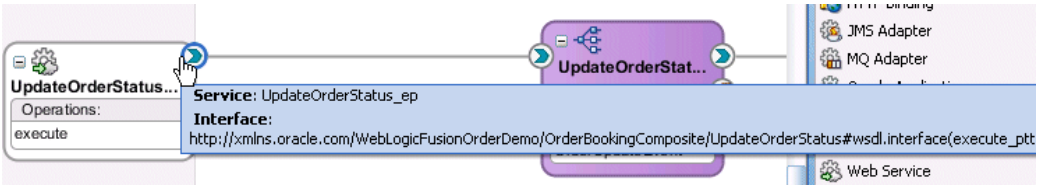
2.3.3 How to View Schemas

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:

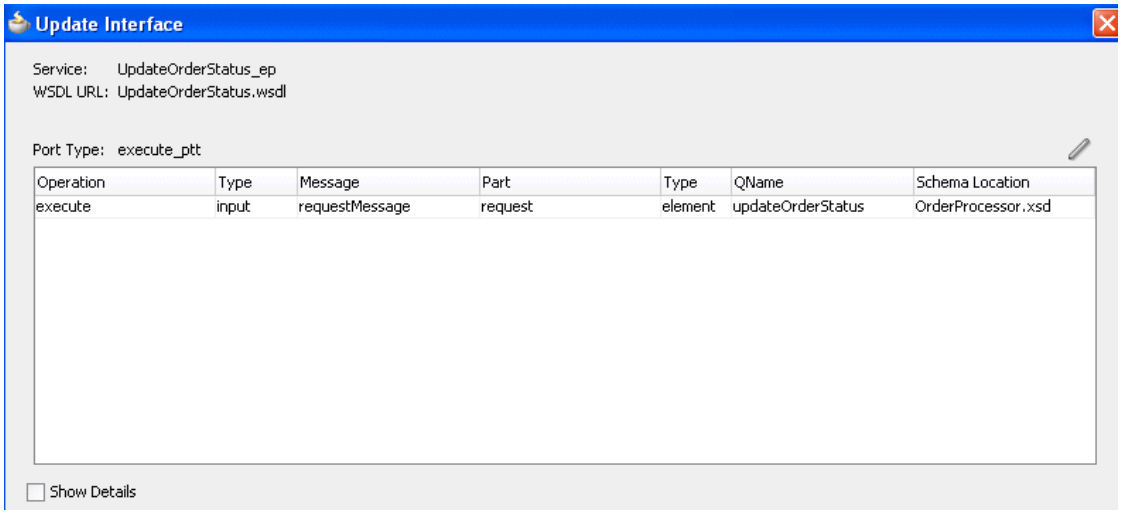
1. Double-click the small arrow handle that appears on the specific binding component or service component. Figure 2–11 provides details.

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

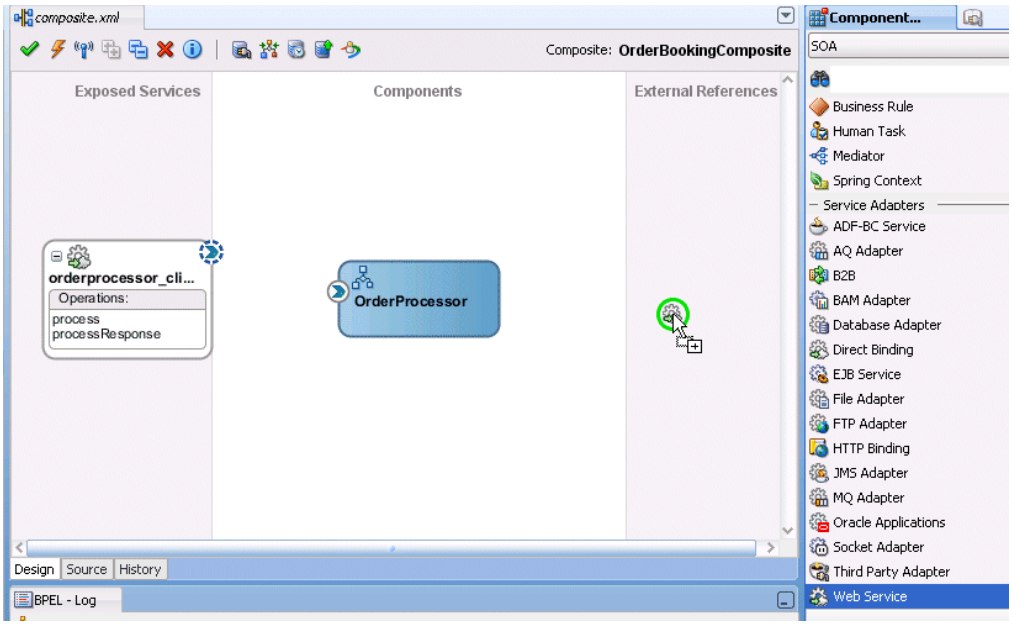
You can use the Component Palette from the SOA Composite Editor to drag and drop reference binding components to the composite.

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.

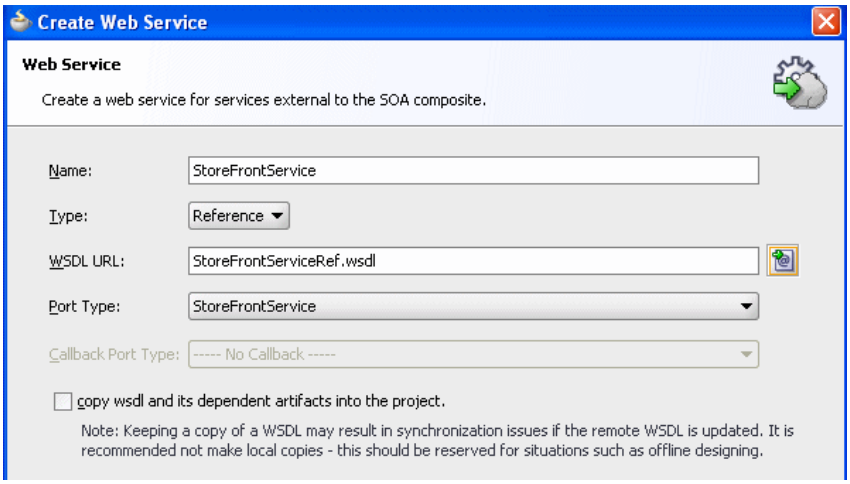
Table 2–10 Reference Editors

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.

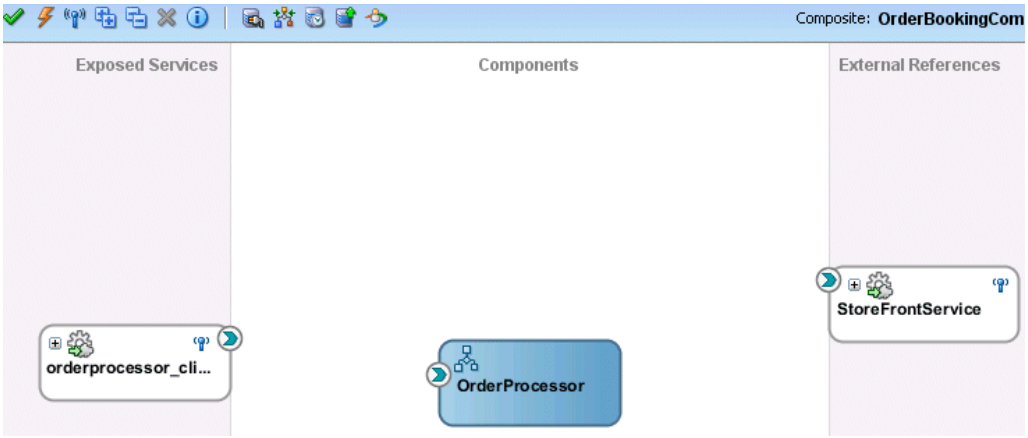
Figure 2–14 Create Web Service Dialog



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 though 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.

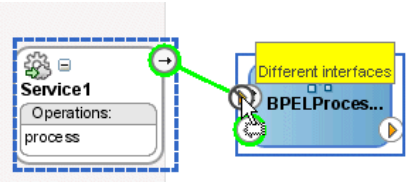
1. In the Create Web Service dialog, click the icon to the right of the **WSDL URL** field to invoke the SOA Resource Browser dialog.
2. Select **Resource Palette** from the list at the top.
3. Expand the nodes under the **Application Server** connection or **WSIL** connection to list all deployed composites 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.

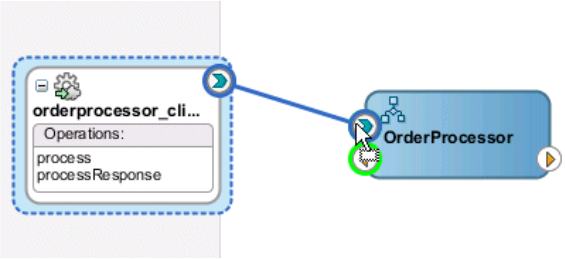
2.5.1 How to Wire a Service and a Service Component

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

To wire a service and a service component:

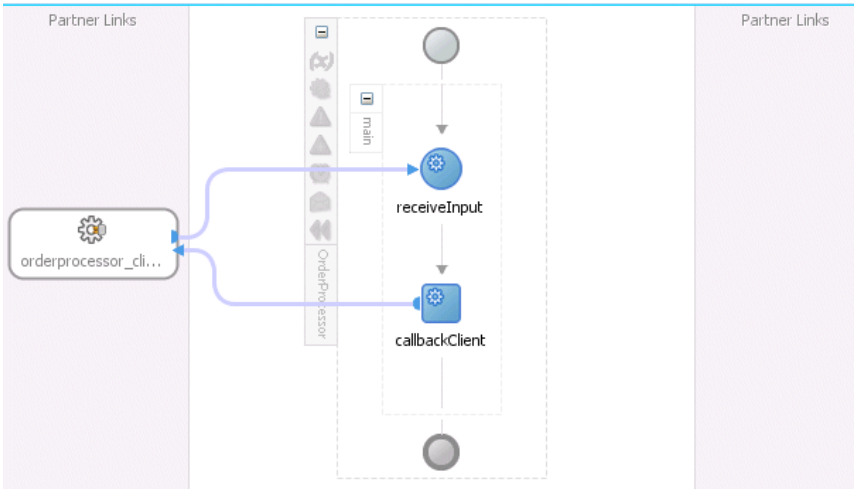
1. From a service reference handle, drag a wire to the service component interface, as shown in Figure 2–17.

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.

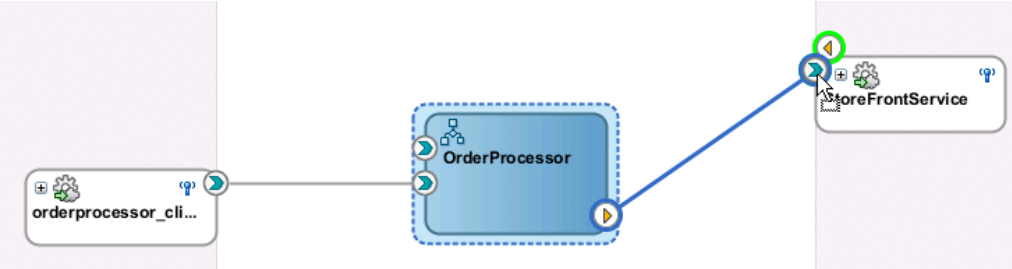
2.5.2 How to Wire a Service Component and a Reference

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

To wire a service component and a reference:

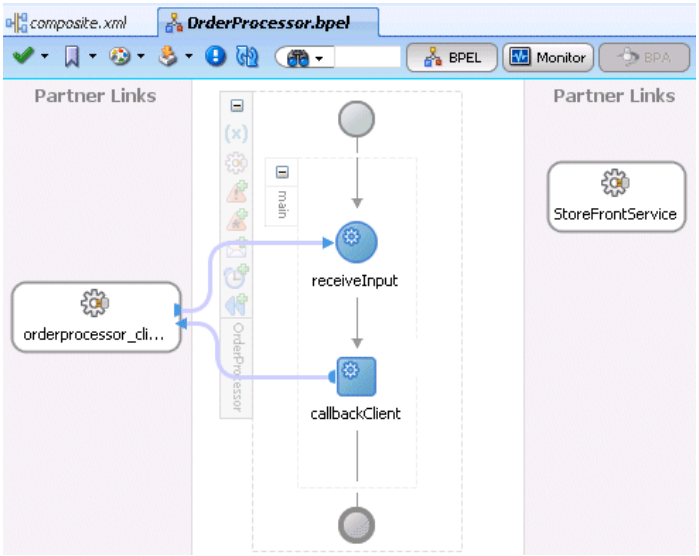
1. In the Application Navigator, double-click **composite.xml** or single-click **composite.xml** above the designer.
2. From the service component, drag a wire to the reference, as shown in Figure 2–19.

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_clientep/OrderProcessorCallback_
pt)"/>
    </callback>
  </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/StoreFron
tService.wsdl">
    <interface.wsdl
 interface="www.globalcompany.example.com#wsdl.interface(StoreFrontService)"/>
    <binding.ws
port="www.globalcompany.example.com#wsdl.endpoint(StoreFrontService/StoreFrontServ
iceSoapHttpPort)"

location="oramds:/apps/FusionOrderDemoShared/services/oracle/fodemo/storefront/sto
re/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."

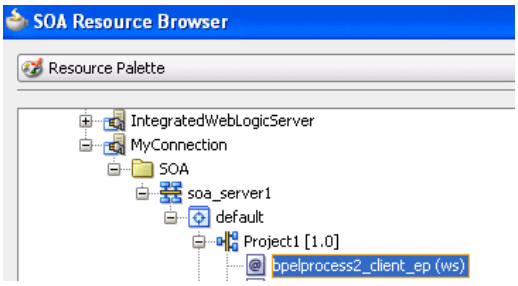
2.7.1 How to Invoke Deployed Composites

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

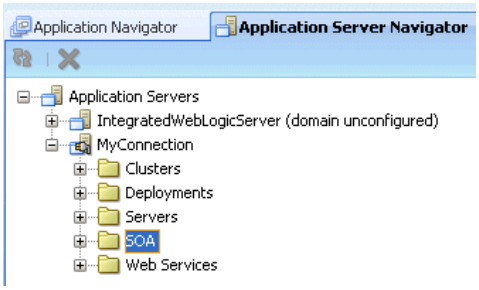
You can manage deployed SOA composite applications from the Application Server Navigator in Oracle JDeveloper. Management tasks consist of undeploying, activating, retiring, turning on, and turning off SOA composite application revisions.

> **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.

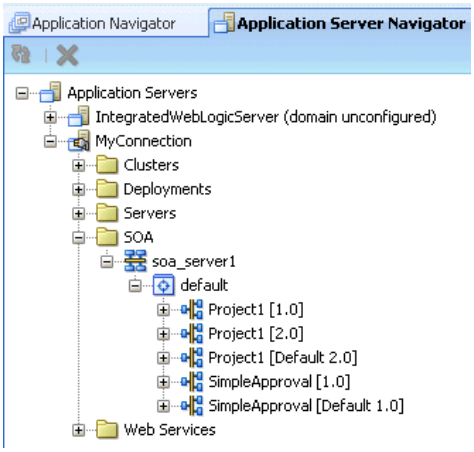
Figure 2–22 Application Server Navigator



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.

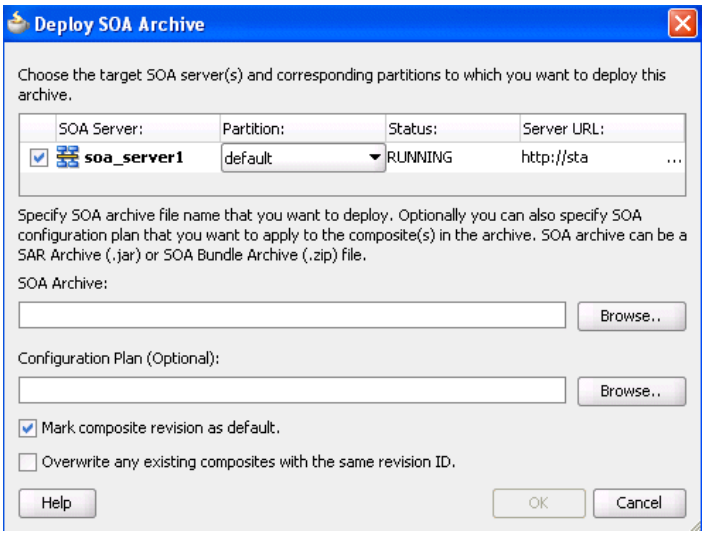
7. If you want to deploy a *prebuilt* SOA composite application archive that includes a deployment profile, right-click the **SOA** folder and select **Deploy SOA Archive**. The archive consists of a JAR file of a single application or a SOA bundle ZIP file containing multiple applications.

   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."

3

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:

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.

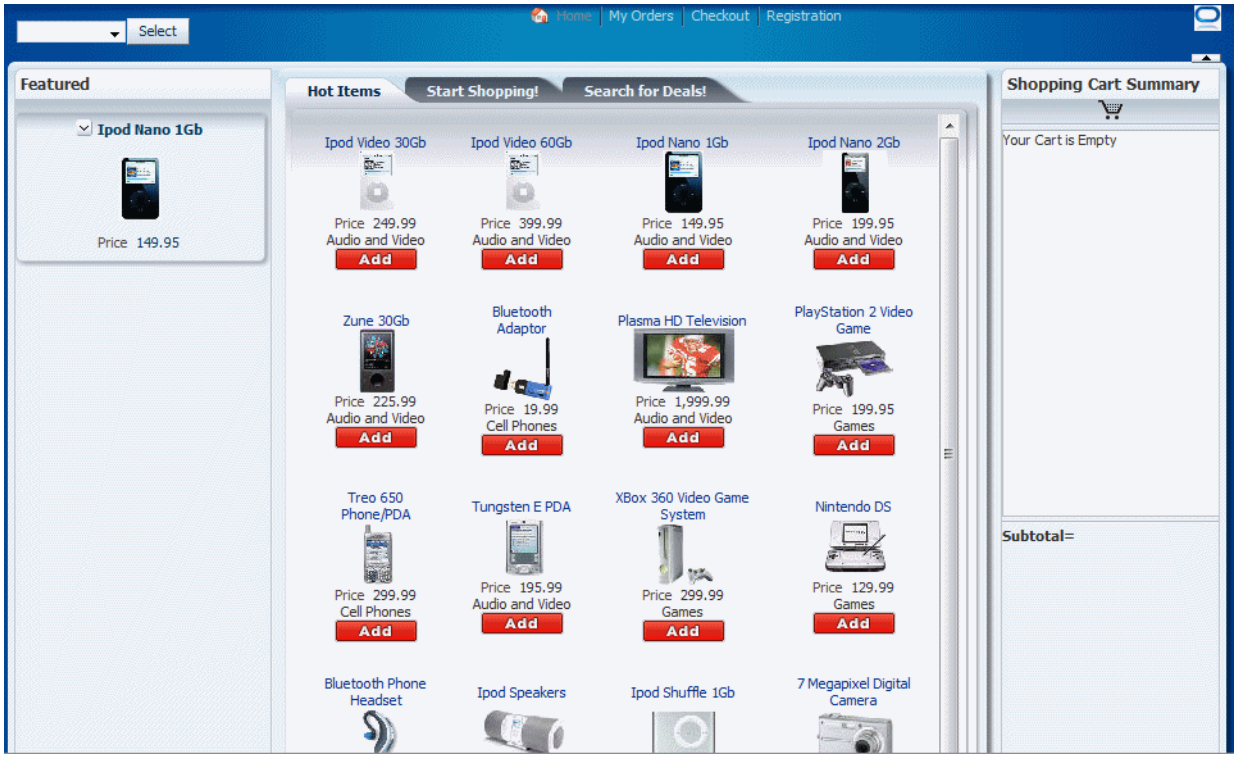
The Store Front module sells electronic devices through a storefront-type web application.

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: his 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 11*g* 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 11*g* 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 and Oracle Business Process Management 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.overwrite.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

`managed_server`	The name of the Managed Server. For example:
	`soa_server1`
	`bam_server1`

`admin_url`	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`.

`username`	The Oracle WebLogic Server administrator. For example: `weblogic`
`password`	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

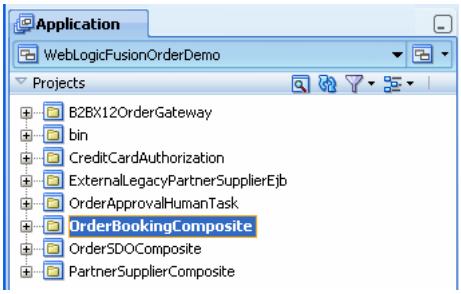
After you have set up the WebLogic Fusion Order Demo application, spend time viewing the WebLogic Fusion Order Demo artifacts in Oracle JDeveloper.

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
B2BX12OrderGateway	This project contains a composite for Oracle B2B. This composite is not used in this guide.
bin	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.
CreditCardAuthorization	This project provides the service needed by **OrderBookingComposite** project to verify the credit card information of a customer.
ExternalLegacyPartnerSupplierEjb	This project provides an external system to provide price quotes.
OrderApprovalHumanTask	This project provides a task form for approving orders from the **OrderBookingComposite** project.
OrderBookingComposite	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.
OrderSDOComposite	This project simulates the **StoreFrontService** service of the Store Front module for testing purposes.
PartnerSupplierComposite	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

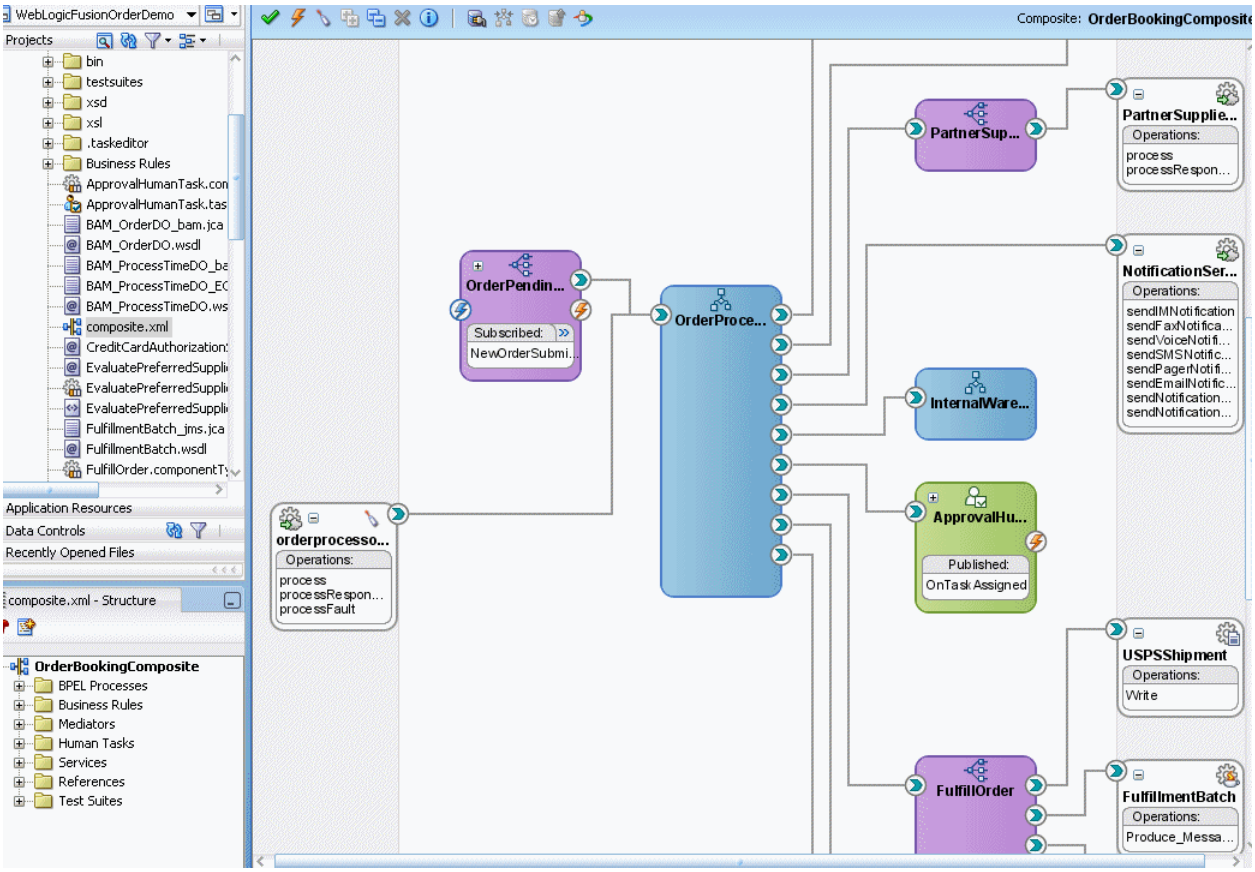
To understand how a composite is designed, examine the main project, OrderBookingComposite, in Oracle JDeveloper.

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



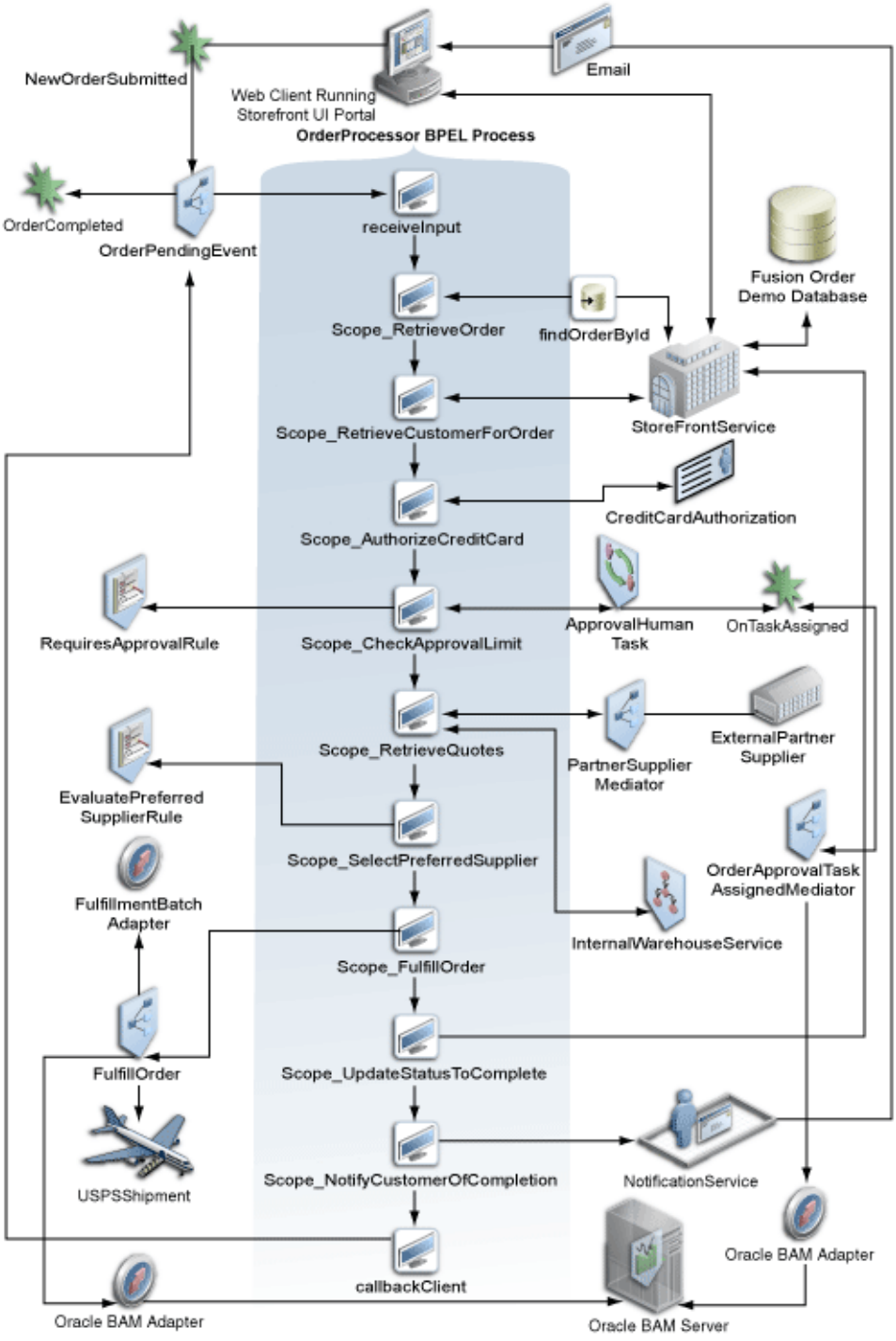
3.4 Understanding the OrderBookingComposite Flow

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

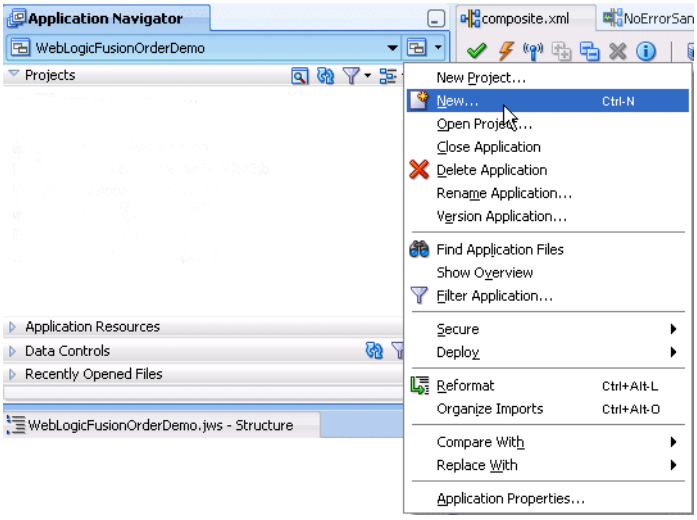
To create a connection to an Oracle WebLogic Server:

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.

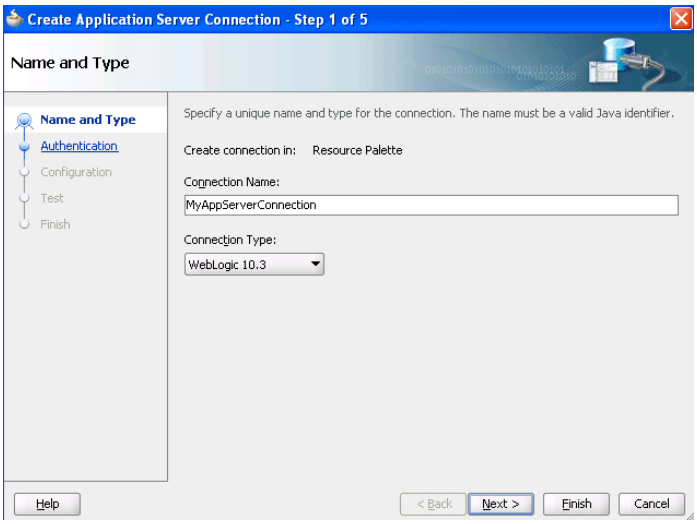
Figure 3–5 Application Menu



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:

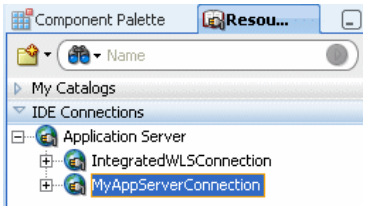
```
Testing JSR-88                          ... success.
Testing JSR-88-LOCAL                    ... success.
Testing JNDI                            ... success.
Testing JSR-160 DomainRuntime           ... success.
Testing JSR-160 Runtime                 ... success.
Testing JSR-160 Edit                    ... 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: `weblogic`
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 11*g*, 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
`jdeveloper.home`	The root directory where you have Oracle JDeveloper 11*g* installed. For example: `C:/JDeveloper/11`
`jdbc.urlBase`	The base JDBC URL for your database in the format `jdbc:oracle:thin:@<`*yourhostname*`>`. For example: `jdbc:oracle:thin:@localhost`
`jdbc.port`	The port for your database. For example: `1521`
`jdbc.sid`	The SID of your database. For example: `ORCL` or `XE`
`db.adminUser`	The administrative user for your database. For example: `system`

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`

7. From the JDeveloper main menu, choose **File > Save All**.
8. In the Application Navigator, under the **Resources** node, right-click **build.xml** and choose **Run Ant Target > buildAll**.
9. 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.

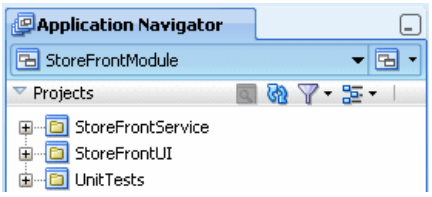
3.5.4 Task 4: Set the Configuration Property for the Store Front Module

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.

1. Choose **File** > **Open**.
2. In the Open dialog, browse to *DEMO_DOWNLOAD_HOME*`/StoreFrontModule` and select **StoreFrontModule.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**.

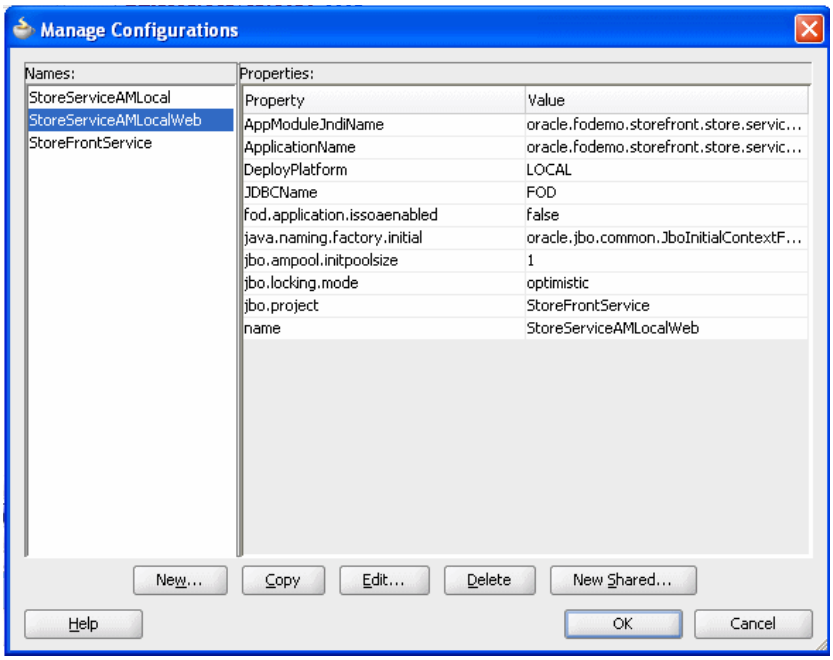
   Figure 3–8 shows the Application Navigator after you open the file for the application workspace.

Figure 3–8 Application Navigator with StoreFrontModule



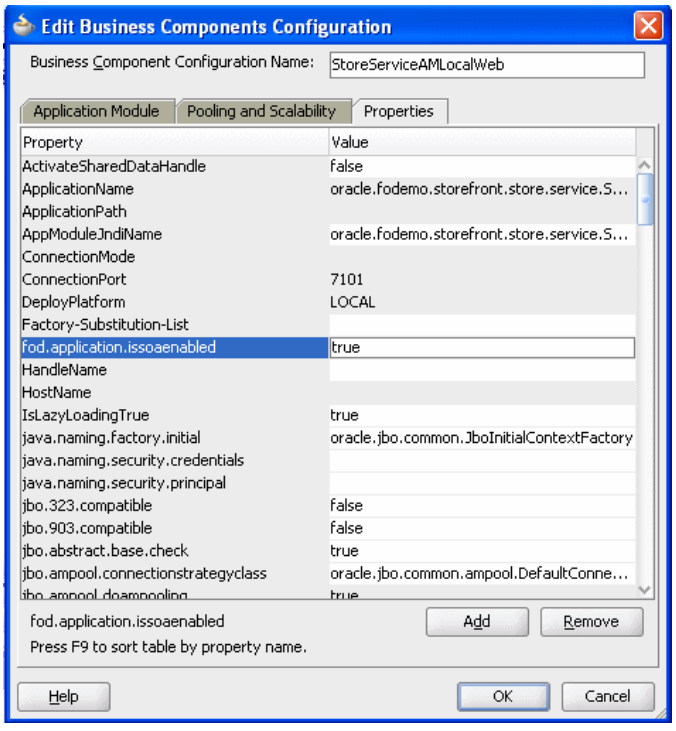
4. In the Application Navigator, expand **StoreFrontService** > **Application Sources** > **oracle.fodemo.storefront** > **store** > **service**.
5. Right-click **StoreServiceAM** and select **Configurations**.
6. In the Manage Configurations dialog, select **StoreServiceAMLocalWeb** in the **Names** 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.issoenabled



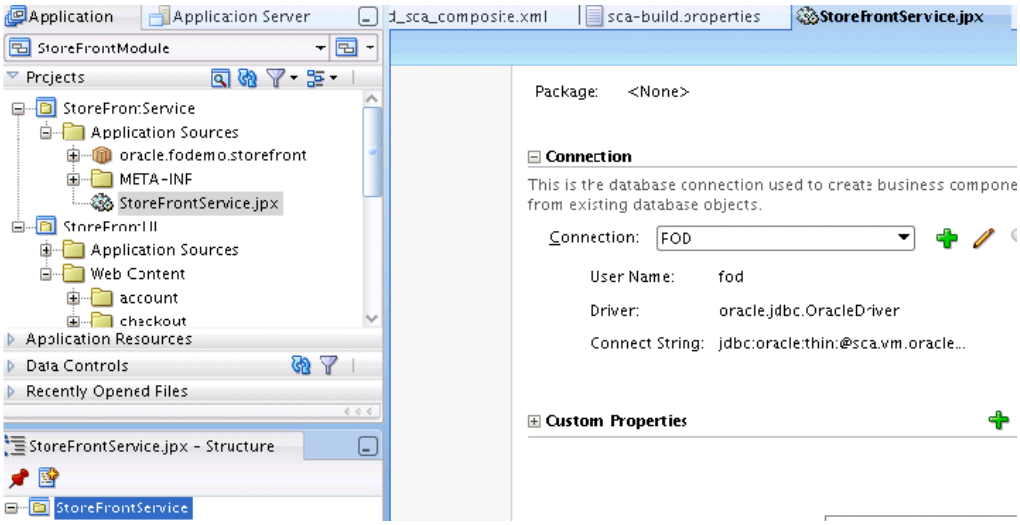
9. Click **OK**.
10. In the Manage Configurations dialog, click **OK**.

3.5.5 Task 5: Edit the Database Connection

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

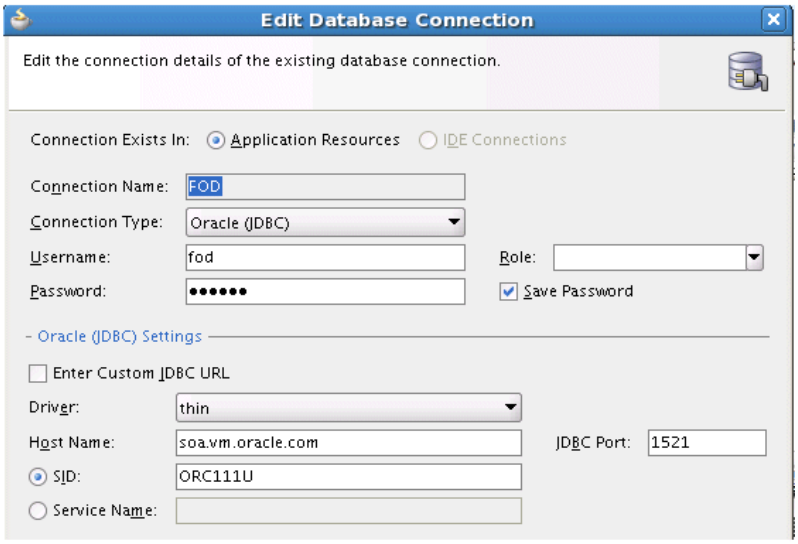
1. In the Application Navigator, expand **StoreFrontService** > **Application Sources**.
2. Double-click **StoreFrontService.jpx**.
3. To the right of the **Connection** field, click the **Edit** icon, as shown in Figure 3–11.

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.

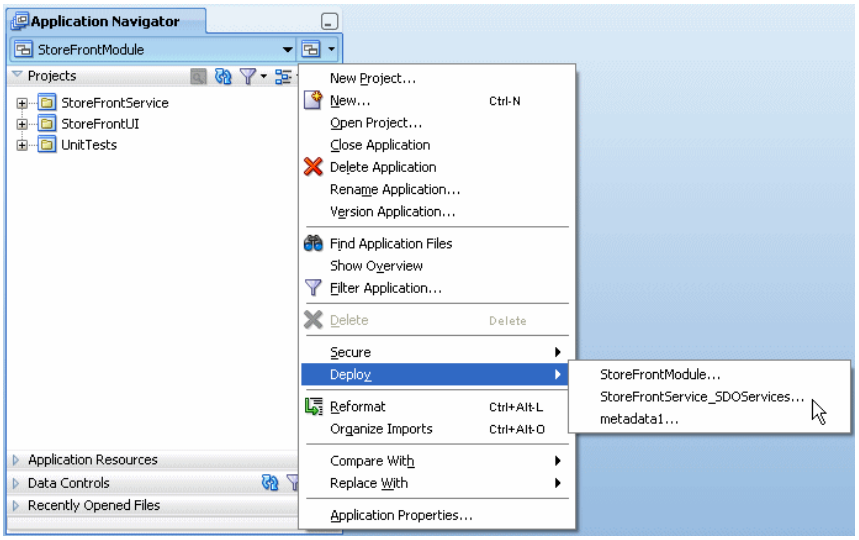
During deployment, Oracle JDeveloper creates the `.jar` and `.war` files and then assembles the `.ear` file, as specified in the deployment profiles. After the file is assembled, Oracle JDeveloper deploys the `.ear` file and unpacks it in a directory on the application server. The directory that is used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

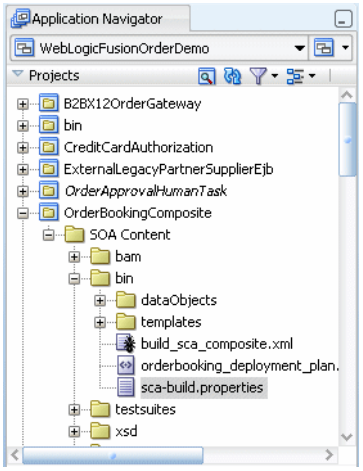
In this task, you deploy the WebLogic Fusion Order Demo application to an Oracle SOA Suite installation, containing an Oracle WebLogic Server domain with an Administration Server and a Managed Server.

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
`enable.bam.sensors`	`true` Set to `true` to enable sensors for Oracle BAM.

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

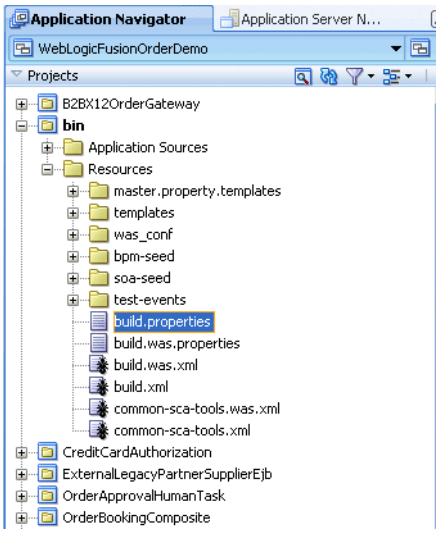
Field	Description
`seed.bam.do`	`true` Set to `true` to seed data objects, alerts, and reports for Oracle BAM. After deployment is done, set this value back to `false`. If this parameter is set to `true` after initial deployment and you redeploy at a later time, then the data objects, alerts, and reports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

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 11*g* 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 `server-setup-seed-deploy-test`.
2. **server-setup-seed-deploy-test**	This script calls the following targets: ■ `compile-deploy-all` compiles, builds, and deploys all the SOA composites to the Managed Server. ■ `seedFodJmsResources` populates the JMS resources for the `Fulfillment` mediator. ■ `seedDemoUsers` adds `jstein` 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 `jstein` 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**.	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:

4

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:

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 *JDev_Oracle_Home*\jdeveloper\JDev\bin\jdev.exe or create a shortcut	$ORACLE_HOME/jdev/bin/jdev

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**.

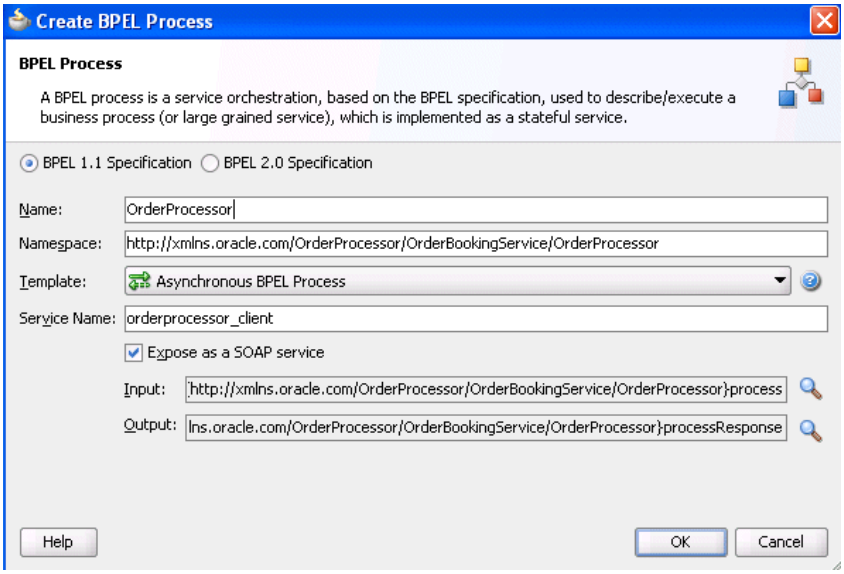
h. Click **Finish**.

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`).

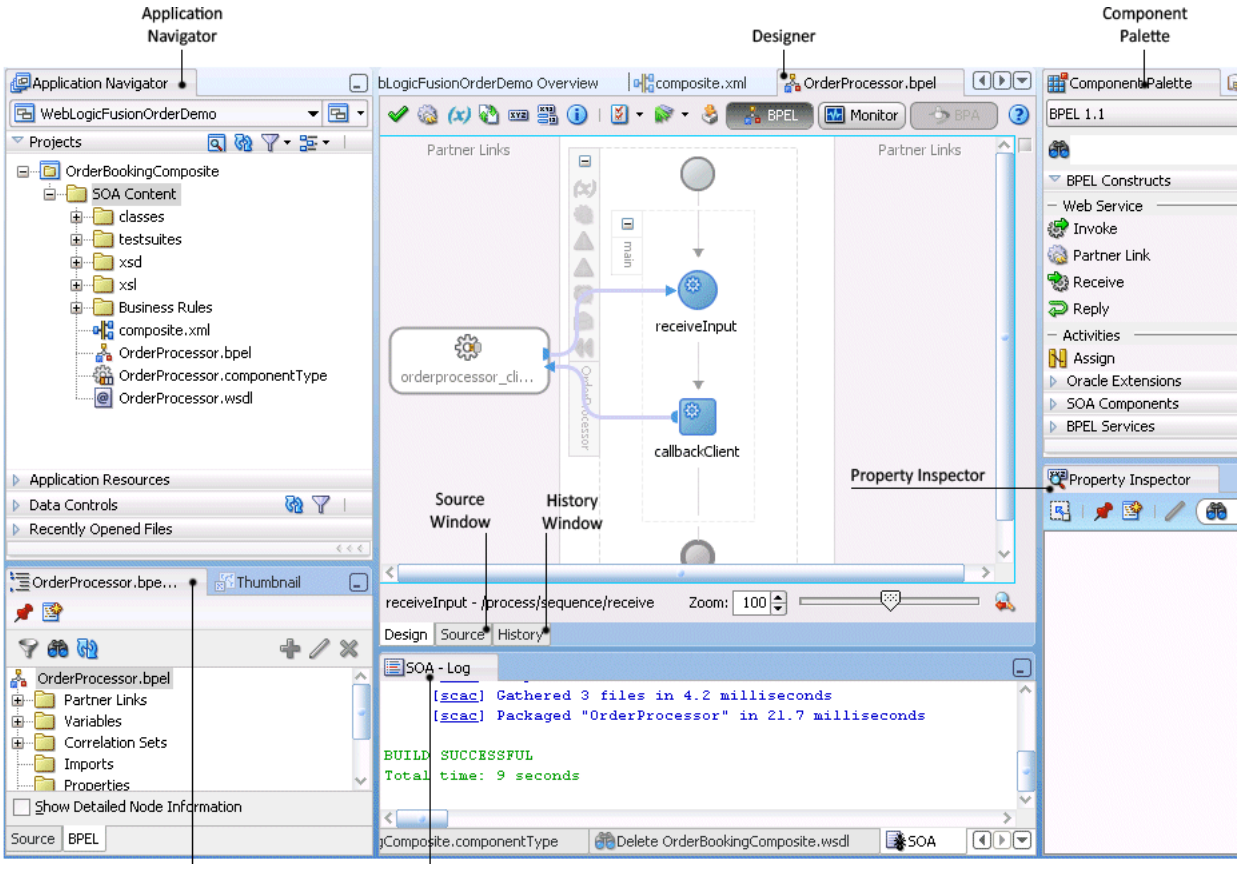
  This is particularly important for business intelligence (BI) data object names, which are generated on the Oracle BAM server in all upper case format. For example, if you create a BPEL process named `BPELProcess1`, a BI name of `BI_DEFAULT_PROJECT1_BPELPROCESS1` is generated for the Oracle BAM BI data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

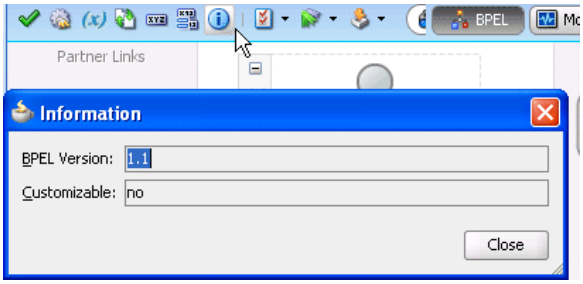
Table 4–2 (Cont.) Oracle JDeveloper Sections

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.

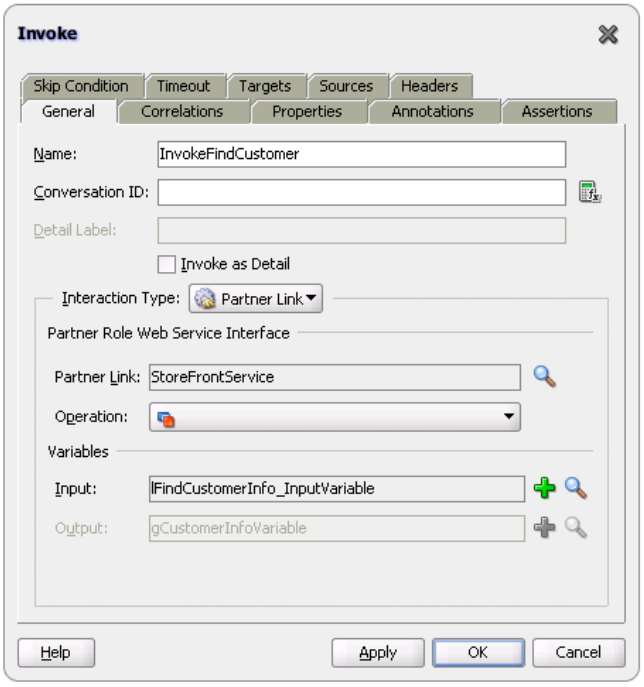
Figure 4–5 Invoke Activity

- 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

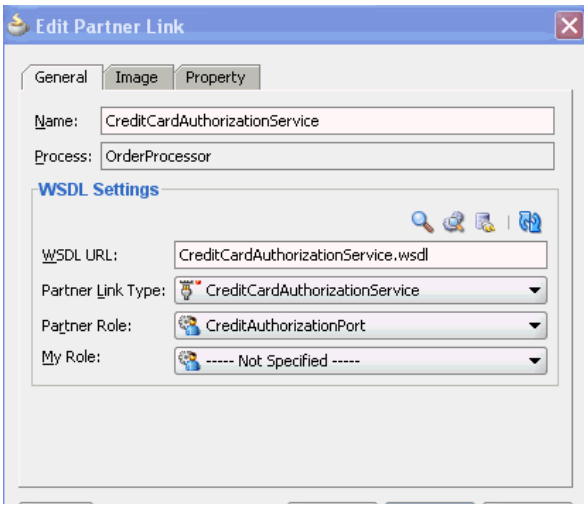


Table 4–3 describes the fields of this dialog.

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:

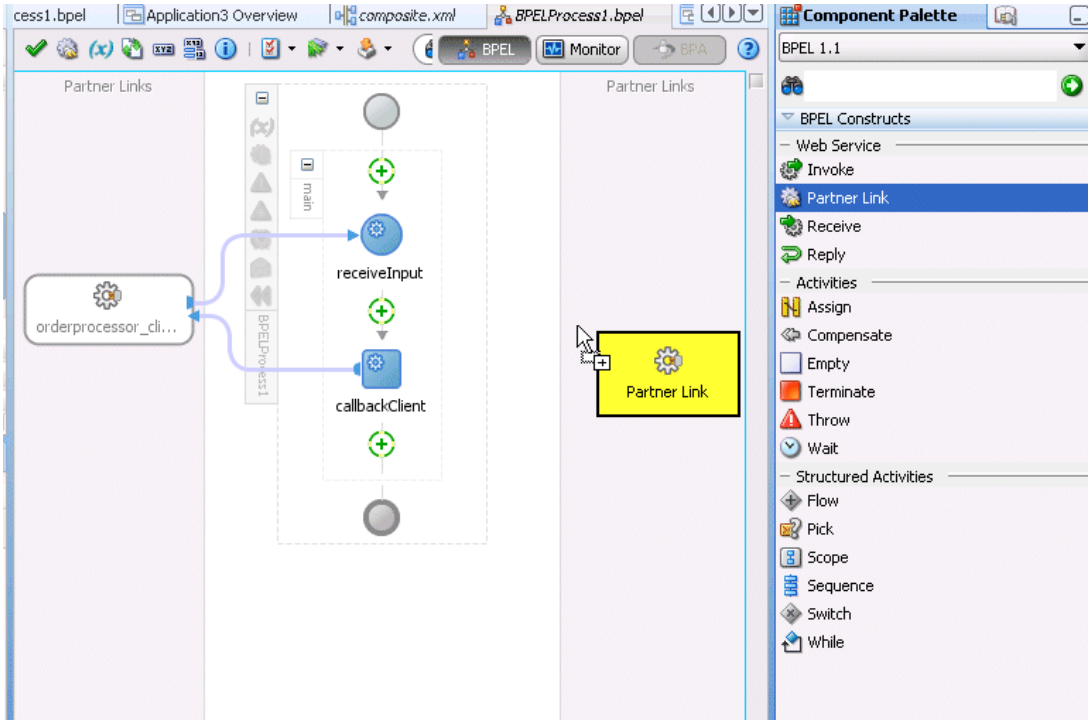
1. In the SOA Composite Editor, double-click the BPEL process service component.

   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

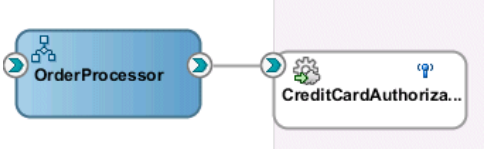
Table 4–4 describes the impact on the SOA Composite Editor.

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 *outbound* 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

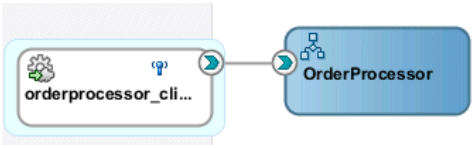
Table 4–5 describes the impact on the SOA Composite Editor.

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 *inbound* 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

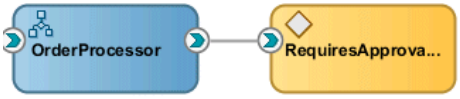
Table 4–8 describes the impact on the SOA Composite Editor.

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

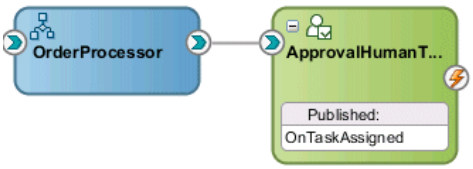
Table 4–9 describes the impact on the SOA Composite Editor.

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



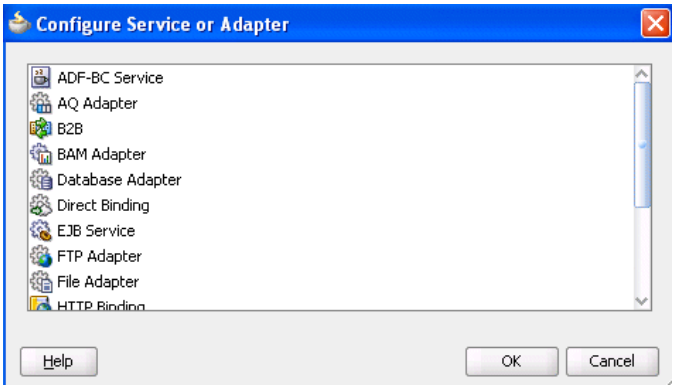
4.5 Introduction to Technology Adapters

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:

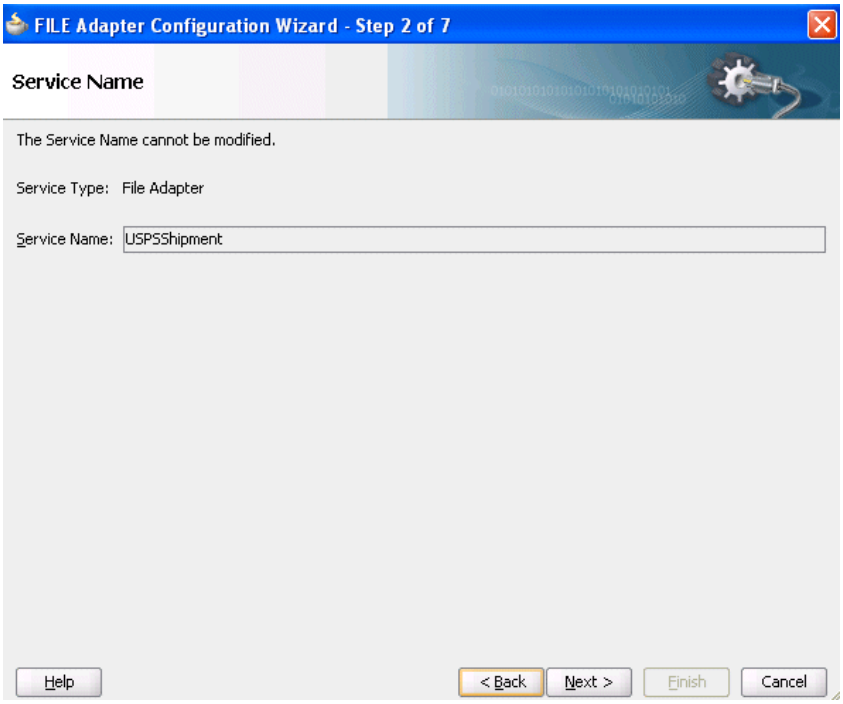
Figure 4–17 Service and Adapter Types



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.

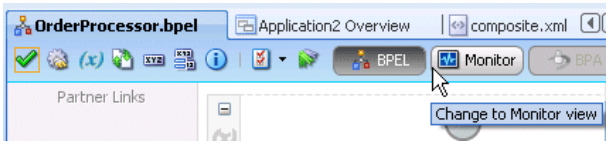
You can also add adapters to your SOA composite application as services or references in the SOA Composite Editor.

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."

5

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:

5.1 Introduction to One-Way Messages

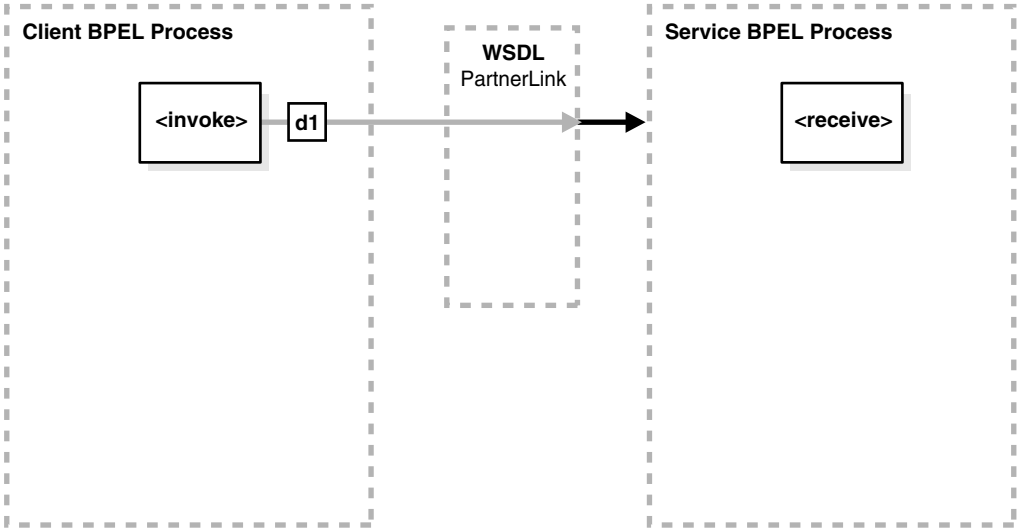
In a one-way message, or fire and forget, the client sends a message to the service (d1 in Figure 5–1), and the service is not required to reply. The client sending the message does not wait for a response, but continues executing immediately. Example 5–1 shows the `portType` and `operation` part of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

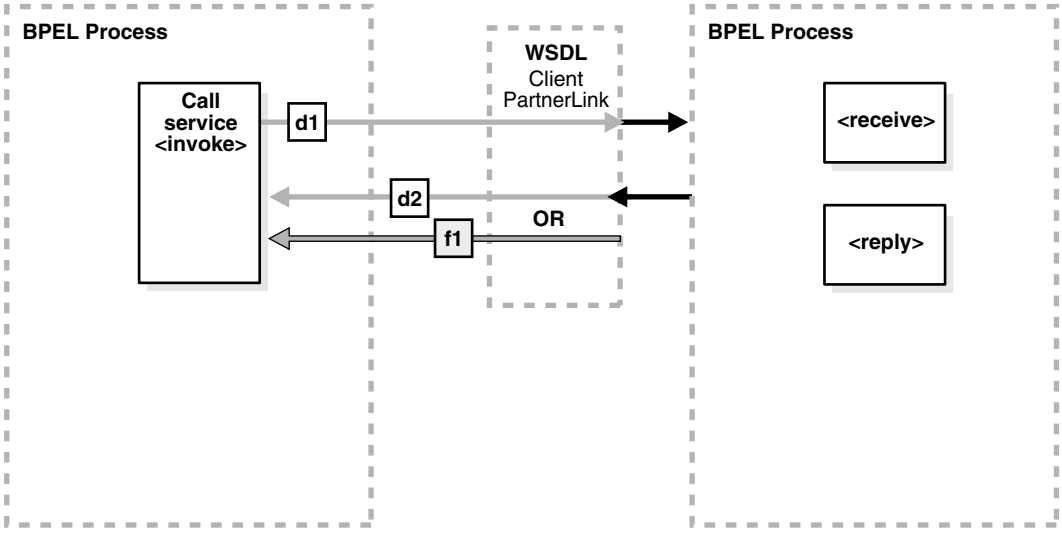
In a synchronous interaction, a client sends a request to a service (d1 in Figure 5–2), and receives an immediate reply (d2 in Figure 5–2). A BPEL process service component can be at either end of this interaction, and must be coded based on its role as either the client or the service. For example, 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

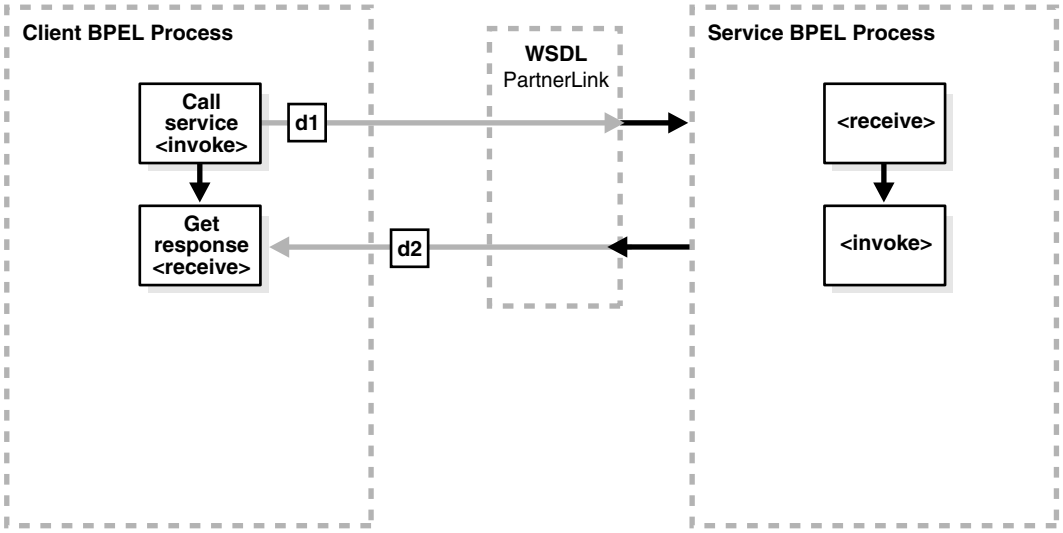
In an asynchronous interaction, a client sends a request to a service and waits until the service replies. Example 5–3 shows the `portType` and `operation` part of the BPEL process WSDL file for this environment.

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

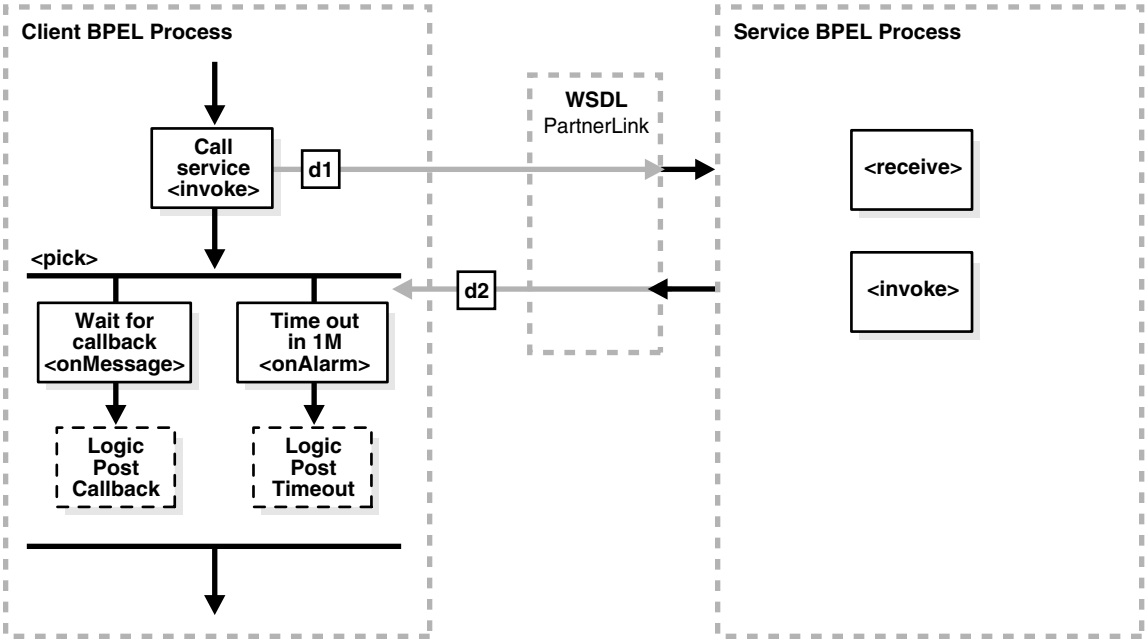
As with a synchronous transaction, when the BPEL process service component is on the service side of an asynchronous transaction, it needs a receive activity to accept the incoming request and an invoke activity to return either the requested information or a fault. Note the difference between this and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

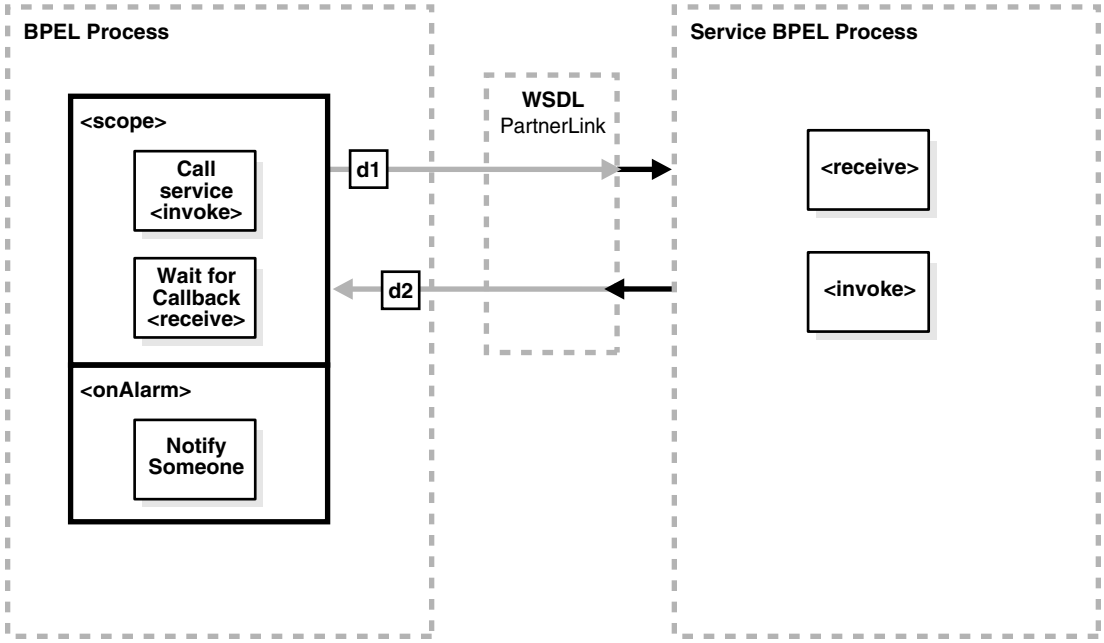
BPEL Process Service Component as the Service

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.

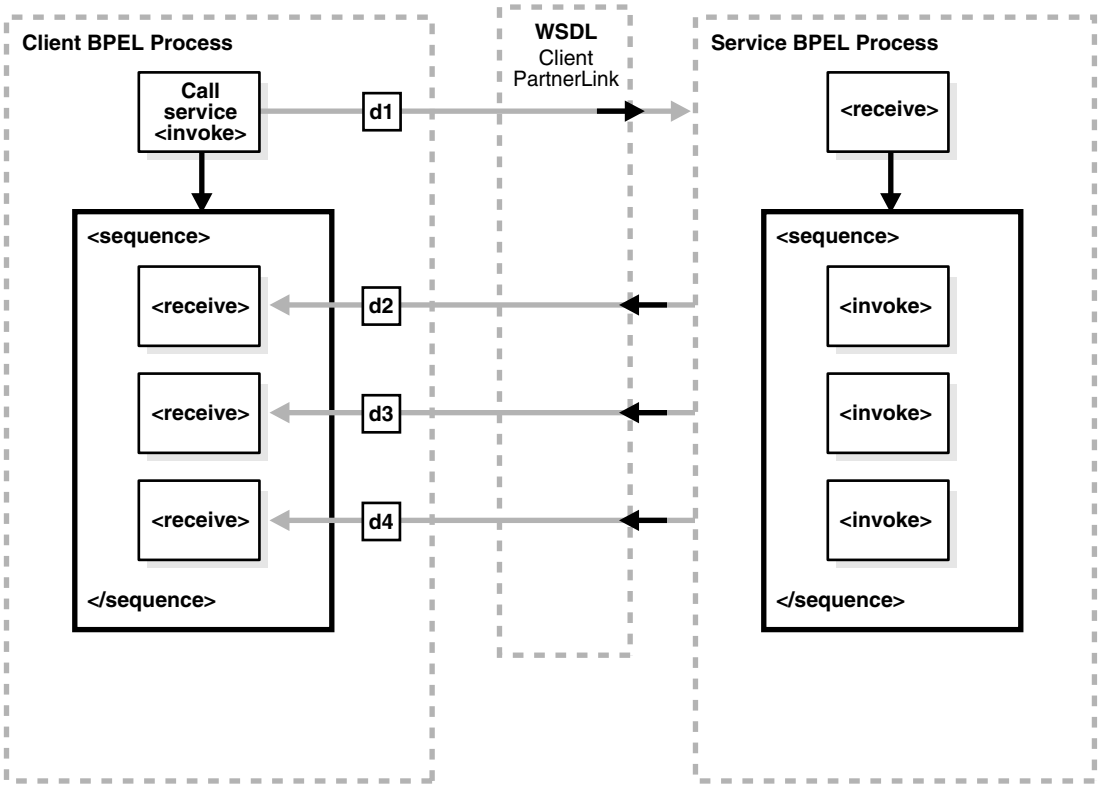
BPEL Process Service Component as the Service

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.

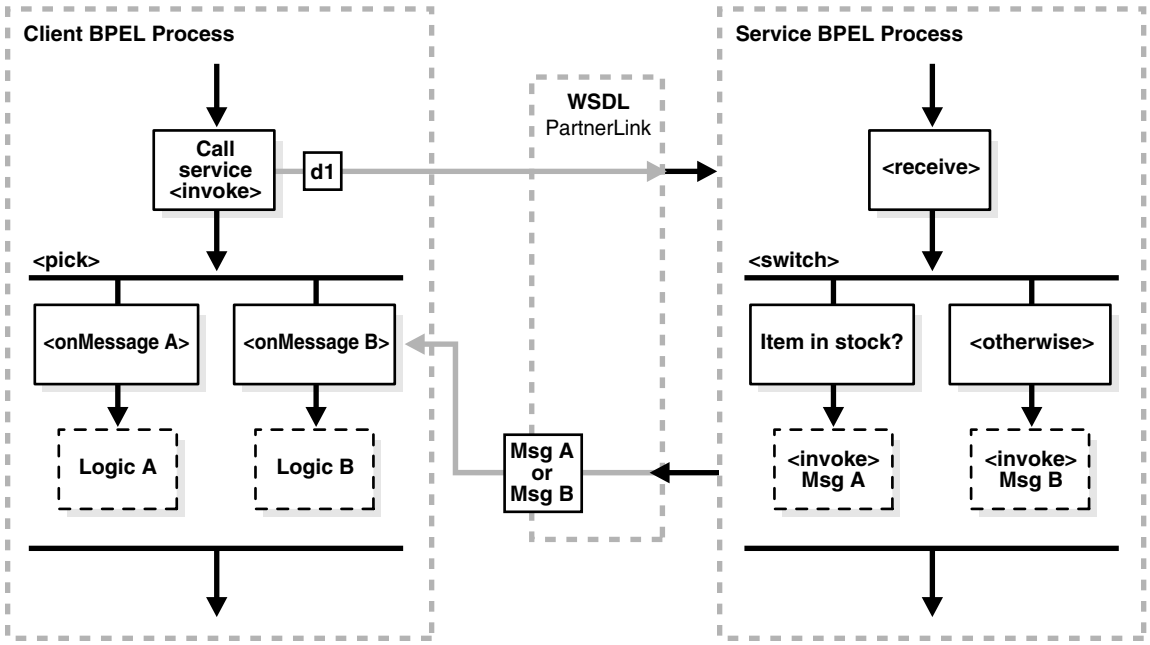
BPEL Process Service Component as the Service

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."

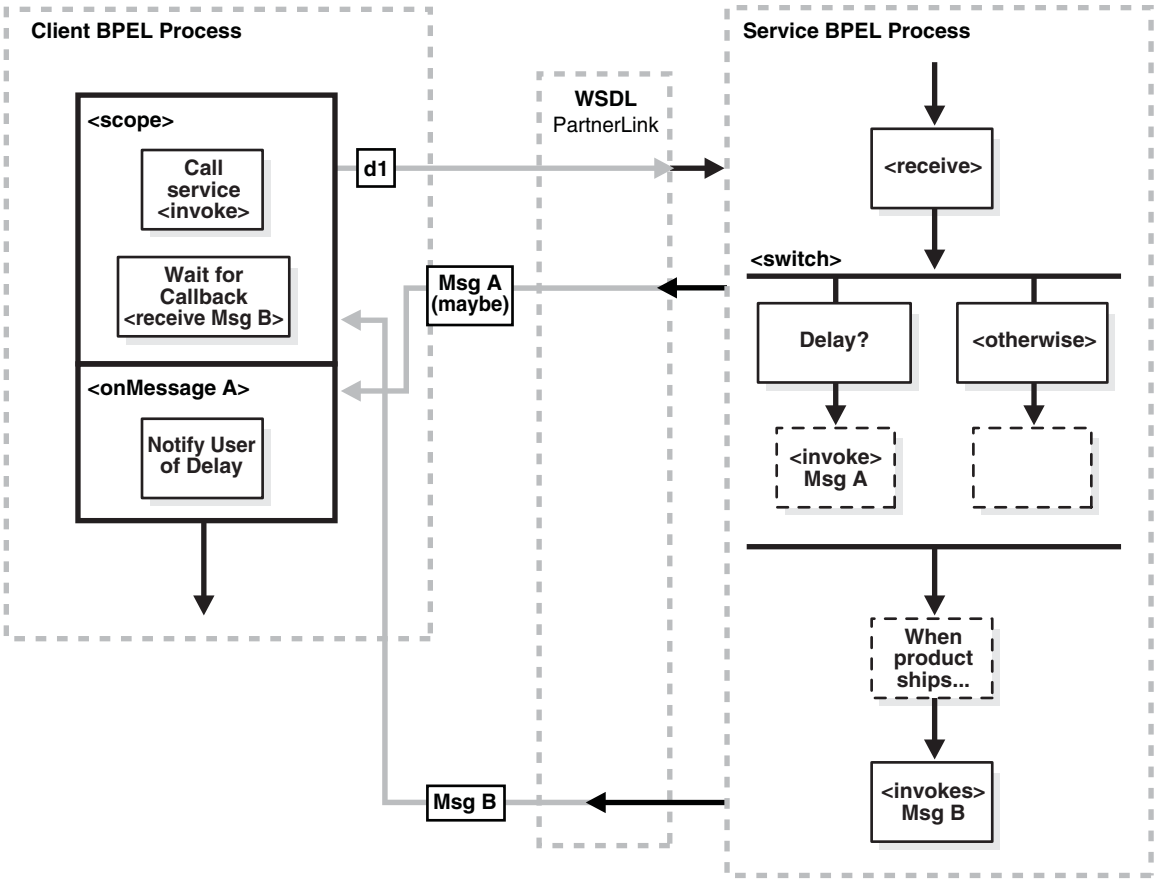
BPEL Process Service Component as the Service

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

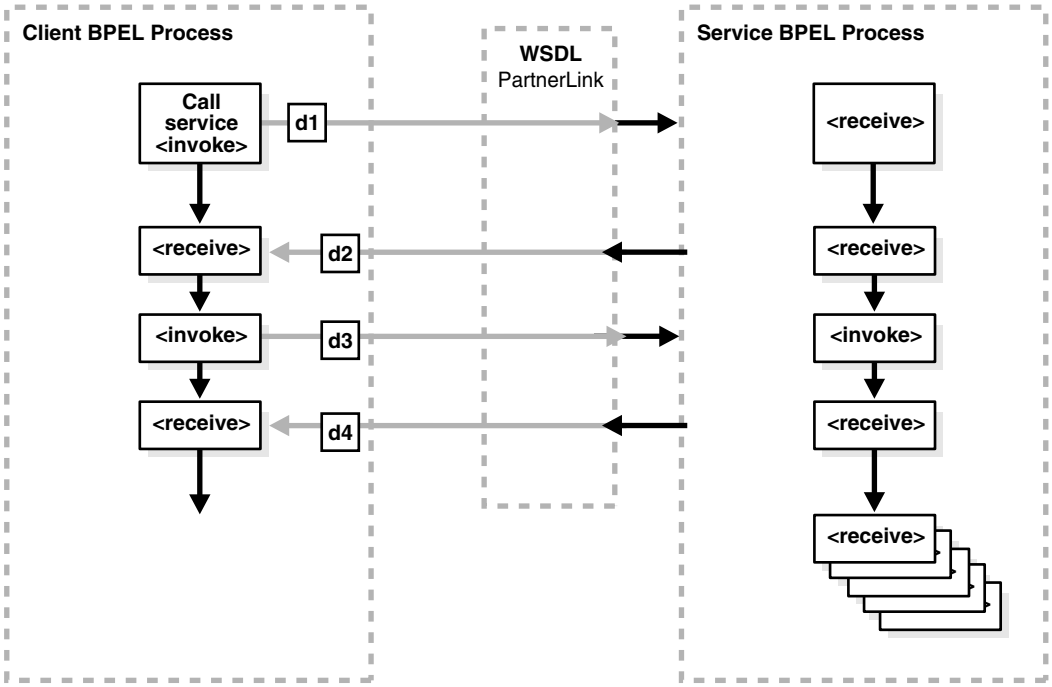
The BPEL service needs a scope activity containing the receive activity and an invoke activity to send the mandatory shipping message, and the scope's onAlarm handler to send the optional delayed message if a timer expires (for example, send the delayed message if the item is not shipped in 24 hours).

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

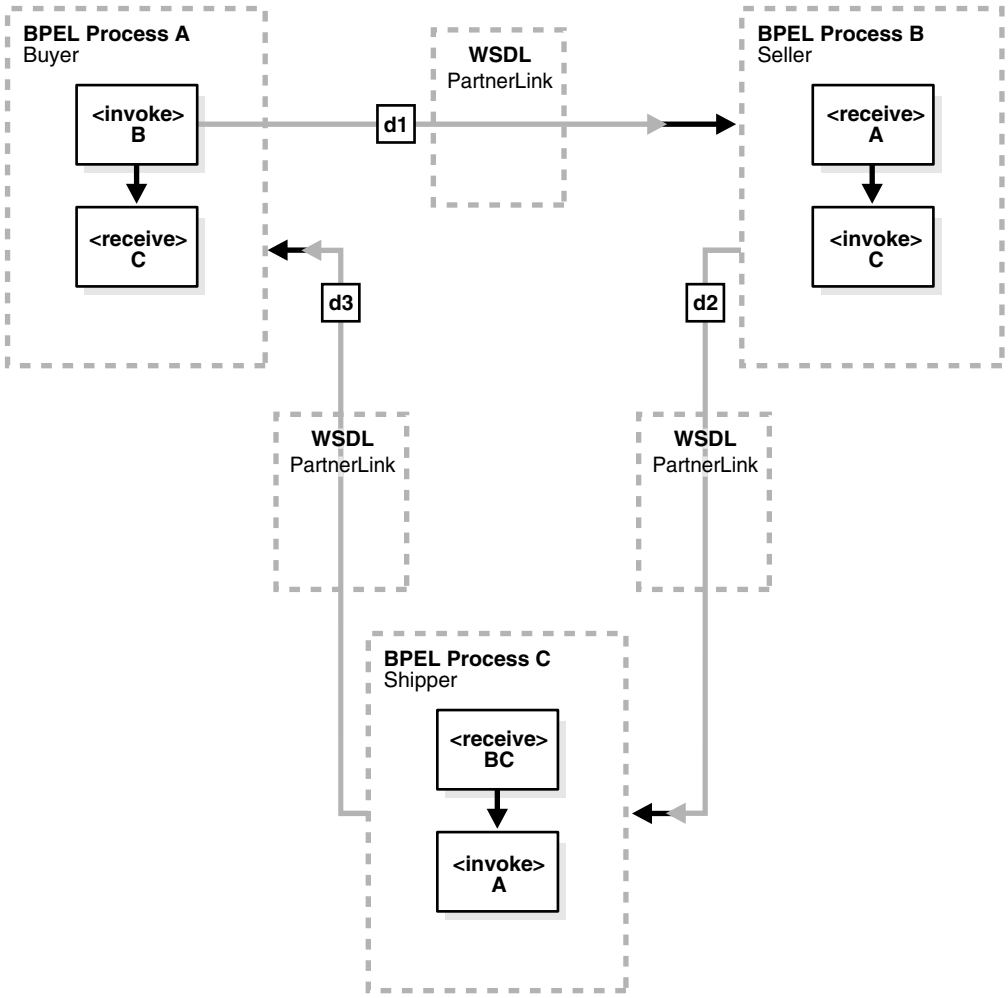
The BPEL service needs a receive activity for each request from the client, and an invoke activity for each response. Once the responses are finished, the BPEL process service component as the service can continue with its processing, using the information gathered in the interaction to perform the necessary 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."

6

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:

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 11*g*, 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 11*g*, 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 11*g* with the entity variable.

Table 6–1 Data Manipulation Capabilities in Previous and Current Releases

10.1.*x* Releases	**11*g* 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 11*g*.

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.

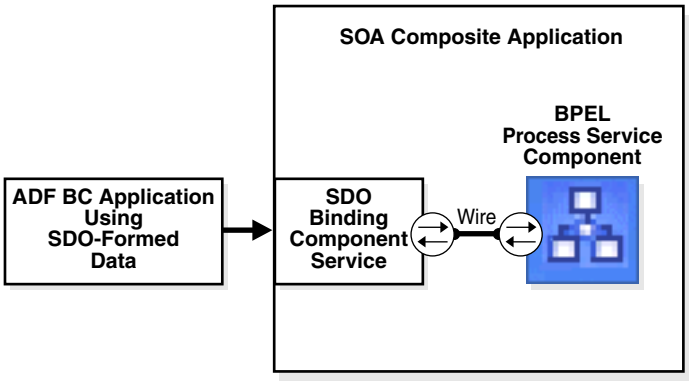
In 10.1.*x* releases of Oracle BPEL Process Manager, all variable data was in DOM form. With release 11*g*, variable data in SDO form is also supported. DOM and SDO variables in BPEL process service components are implicitly converted to the required forms. For example, an Oracle BPEL process service 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:

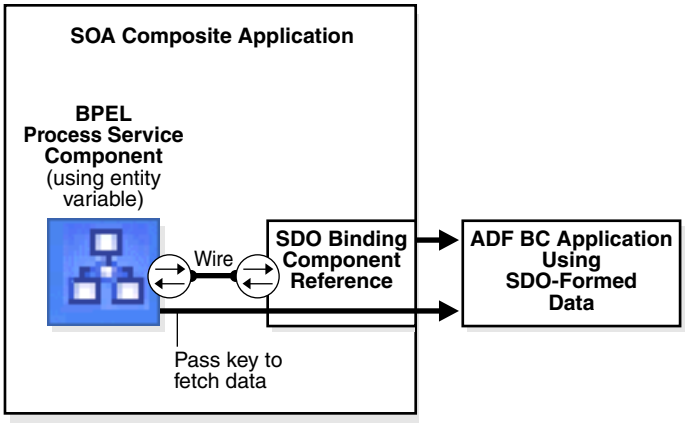
- Define an SDO binding component service and a BPEL process service component in the composite application.
- Connect (wire) the SDO service and BPEL process service component.
- Define the details of the BPEL process service component.

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.

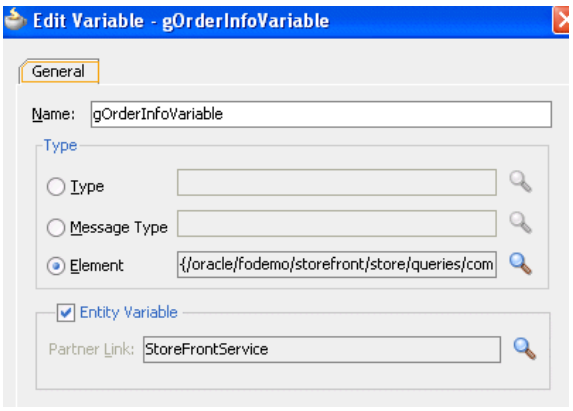
2. Right-click the **Variables** folder and select **Expand All Child Nodes**.
3. In the second **Variables** folder, right-click and select **Create Variable**.

   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.

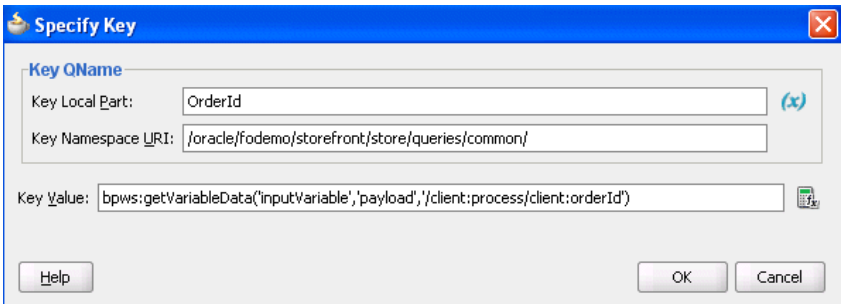
   The Specify Key dialog appears. You use this dialog to create a key for retrieving the order ID from the Oracle ADF Business Component data provider service.
7. Enter the details described in Table 6–2 to define the binding key:

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: ▪ `$inputMsg.payload/tns:poid` ▪ `bpws:getVariableData('inputmsg','payload','tns:poid')` 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.

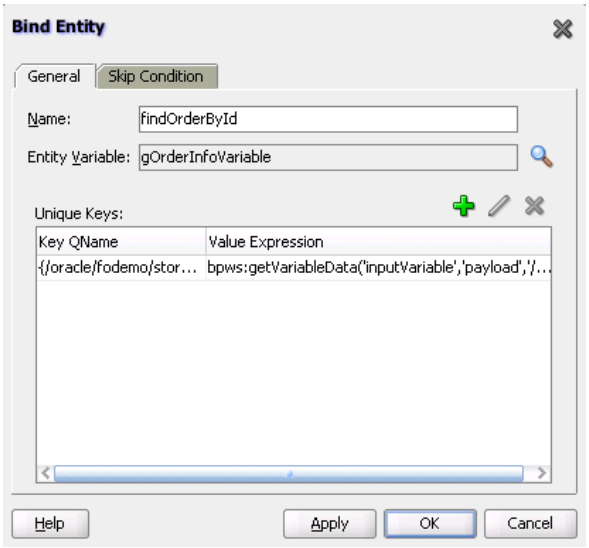
Figure 6–4 Specify Key Dialog



8. 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

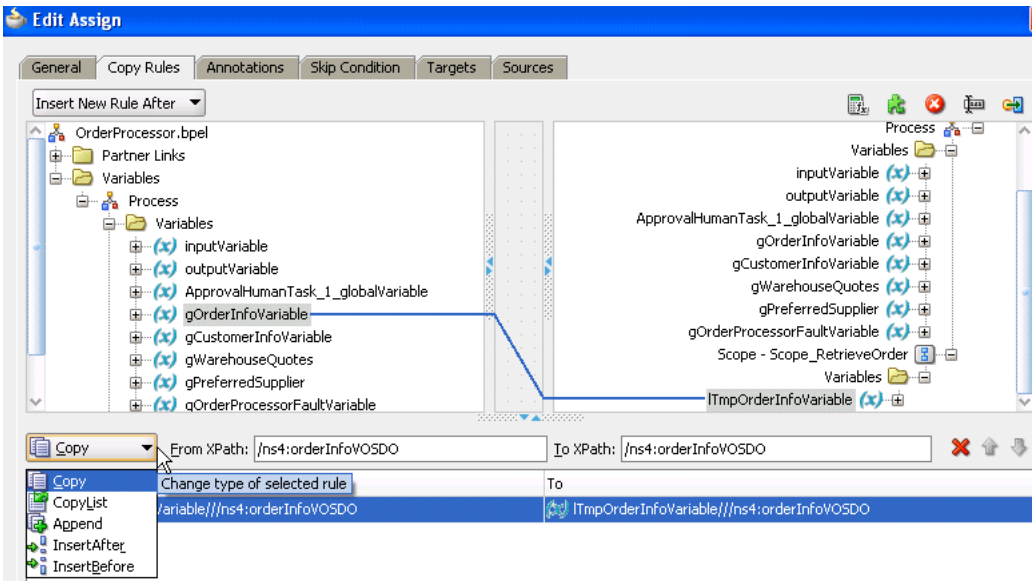
```
<complexType name="CustomerProfileType">
   <sequence>
      <element name="name" type="string"/>
      ...
   </sequence>
   <attribute name="custId" type="string"/>
</complexType>
```

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.

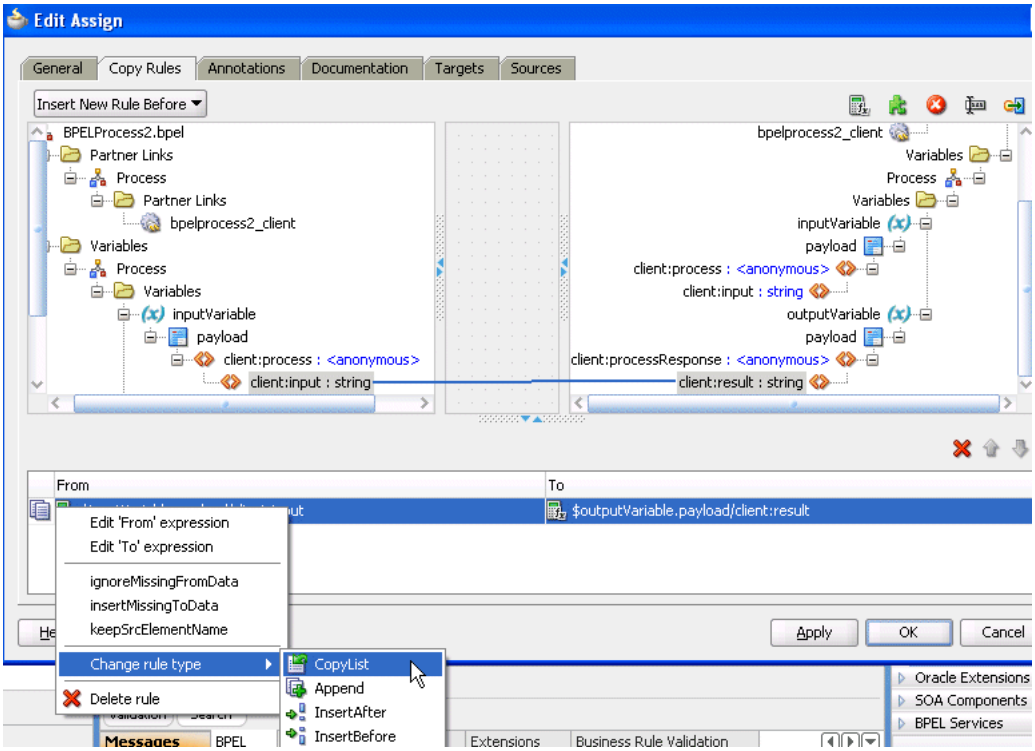
In BPEL 1.1, you select an extension type from the dropdown list, as shown in Figure 6–6.

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.

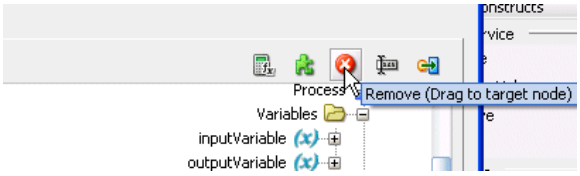
Example 6–52 bpelx:insertAfter Extension in BPEL 2.0

```
<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

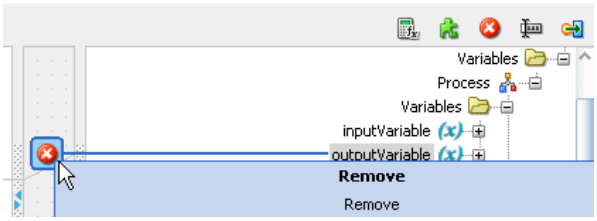
The `bpelx:remove` extension in an assign activity enables a BPEL process service component to remove a variable. In Oracle BPEL Designer, you add the `bpelx:remove` extension by dragging the **remove** icon in the upper right corner of the **Copy Rules** tab to the target variable you want to remove, 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>
        <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>
```

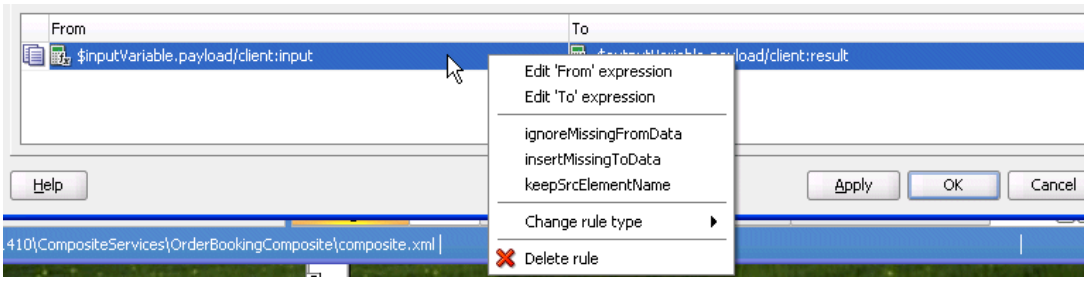
6.14.7 How to Use Assign Extension Attributes

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
`<copy bpelx:ignoreMissingFromData="yes|no"/>`	`<copy ignoreMissingFromData="yes|no"/>`

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
`<copy bpelx:insertMissingToData="yes|no"/>`	`<copy bpelx:insertMissingToData="yes|no"/>`

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	`<copy keepSrcElementName="yes|no"/>`

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`

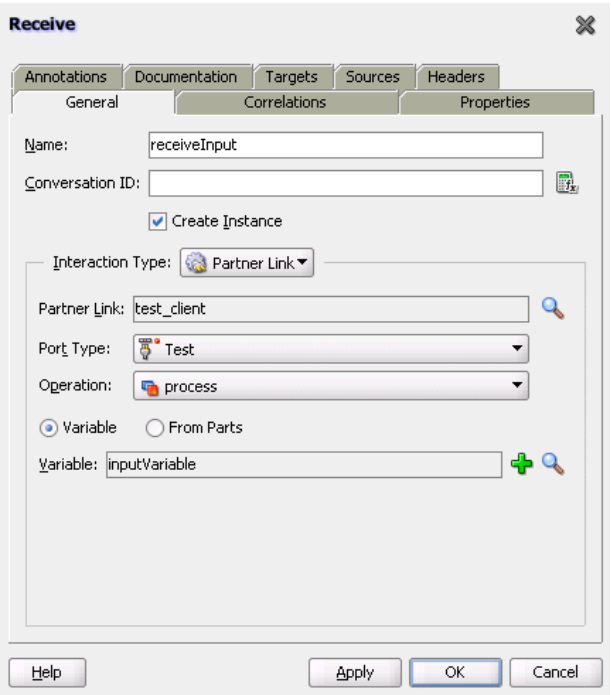
6.17.1 How to Map WSDL Message Parts

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

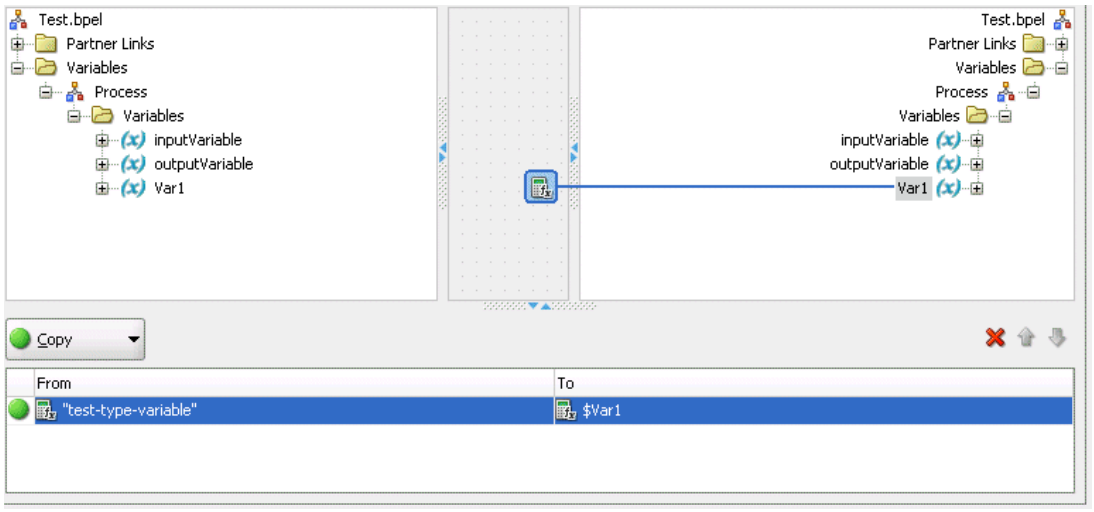
1. Note the receive activity in Figure 6–11 includes a standard **inputVariable** variable from the client.

Figure 6–11 Receive Activity



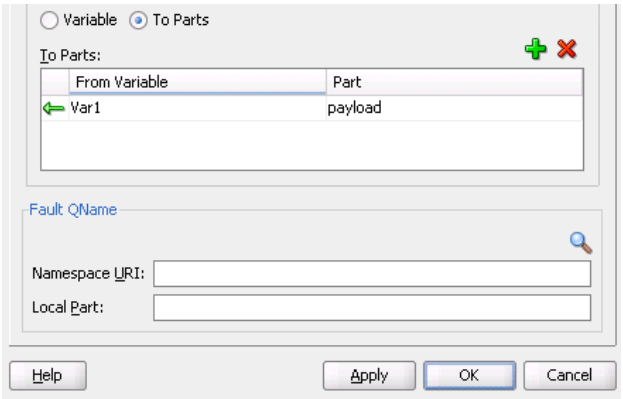
2. Note the assign activity in Figure 6–12 copies the **test-type-variable** contents to **Var1**.

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>
  <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 *nth* 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 `bwps: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 `bwps: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 `bwps: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&gt;2000
            &lt;/lineTotal&gt;
            &lt;/item&gt;')"/>
         <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 `.bpel` 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>
```

7

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 (`bpel-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.

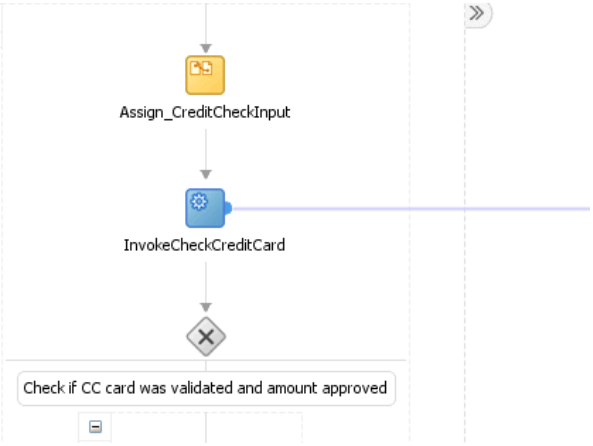
7.2.1 How to Invoke a Synchronous Web Service

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

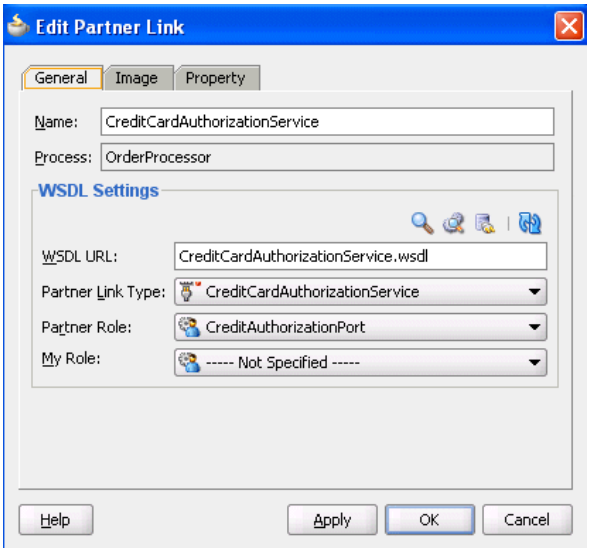
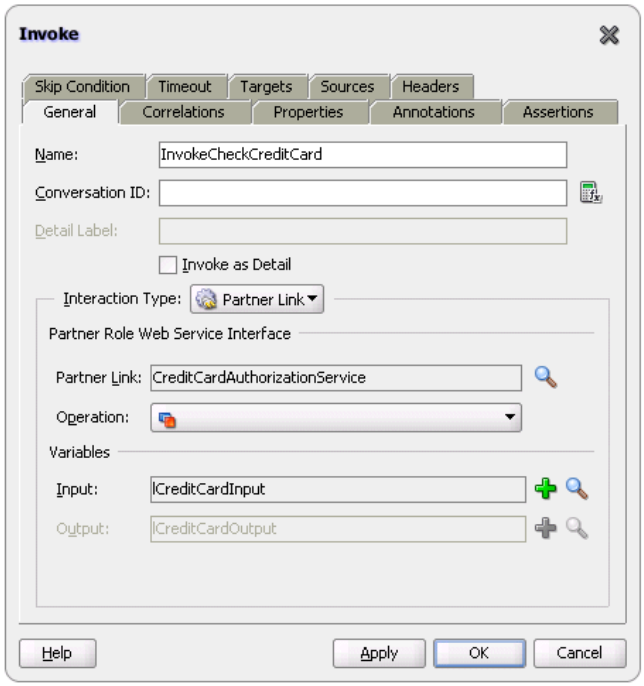


Figure 7–3 shows the **InvokeCheckCreditCard** invoke activity.

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`.

8

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:

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.

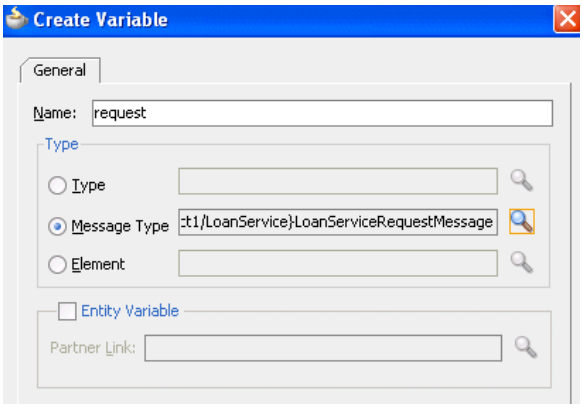
To display the message type, select the **Message Type** option, and then select its **Browse** icon to display the Type Chooser dialog. From here, expand the Message Types tree to make your selection. For this example, **Message Types** > **Partner Links** > **Loan Service** > **LoanService.wsdl** > **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

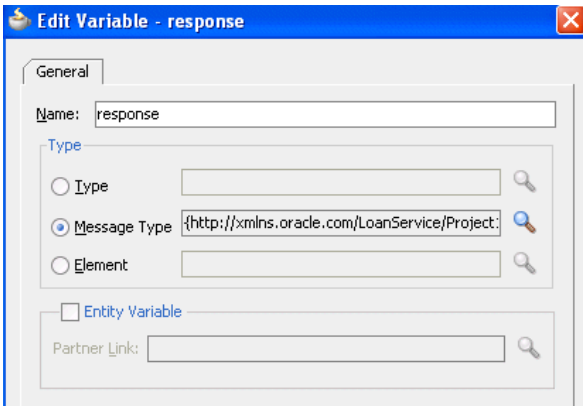
Follow these steps to create a receive activity and a global output variable named `response`. This activity waits for the loan application approver web service's callback operation. The loan application approver web service uses this output variable to send the loan offer result to the client.

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

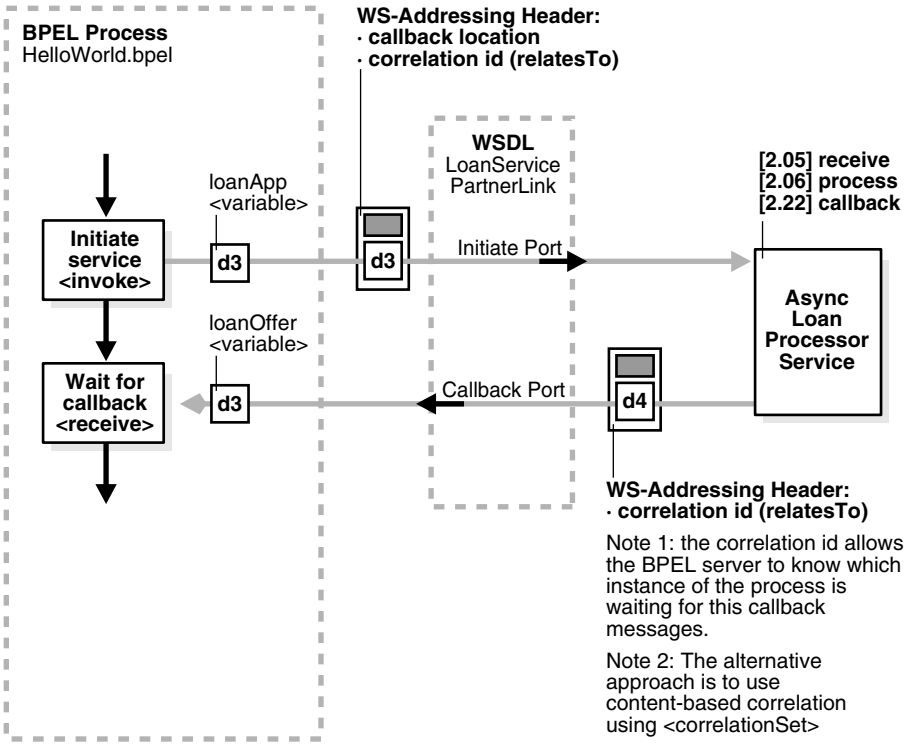


Figure 8–3 shows how messages are passed along with WS headers so that the response can be sent to the correct destination.

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.SCAReference.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	***directory_path*/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	***directory_path*/SecondFileRead.wsdl**
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	*directory_path*/**ThirdFileRead.wsdl**
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 *not* 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 *not* 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** > **LoanAppl_msg** > **Part - LoanAppl**.
5. In the **Query** field, press Ctrl+Space to define the following XPath expression:

   ```
   /ns2:LoanAppl/ns2:Name
   ```

6. Click **OK**.
7. Repeat Step 1 through Step 3 to create a second property alias for **NameCorr**.
8. Expand and select **Message Types** > **Project WSDL Files** > **SecondFileRead.wsdl** > **Message Types** > **LoanAppResponse_msg** > **Part - LoanAppResponse**.
9. In the **Query** field, press Ctrl+Space to define the following XPath expression:

   ```
   /ns4:LoanAppResponse/ns4:APR
   ```

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."

9

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"

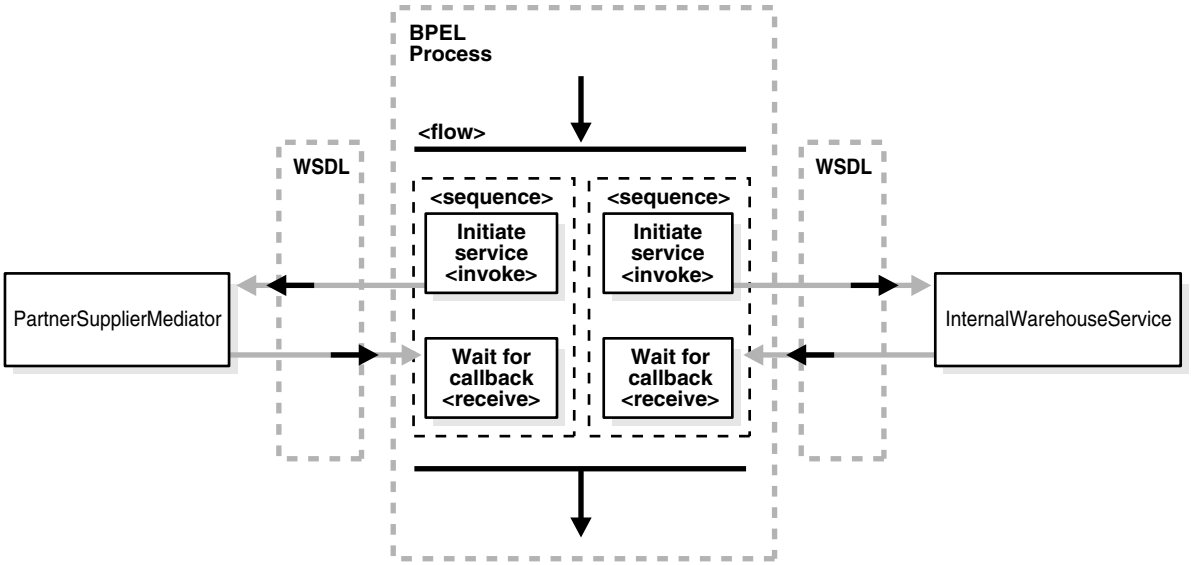
For additional information on creating parallel flows in a SOA composite application, see the Fusion Order Demo application, which is described in Chapter 3, "Introduction to the SOA Sample Application."

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

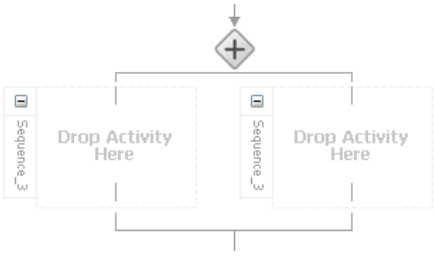
You can create a parallel flow in a BPEL process service component with the flow activity. The flow activity enables you to specify one or more activities to be performed concurrently. The flow activity also provides synchronization. The flow activity completes when all activities in the flow have finished 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.

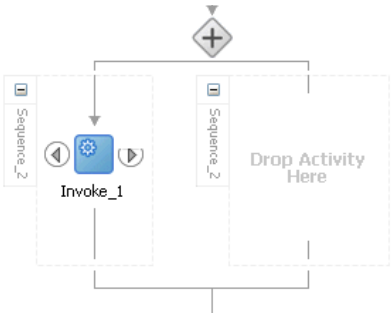
Figure 9–2 Flow Activity



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.

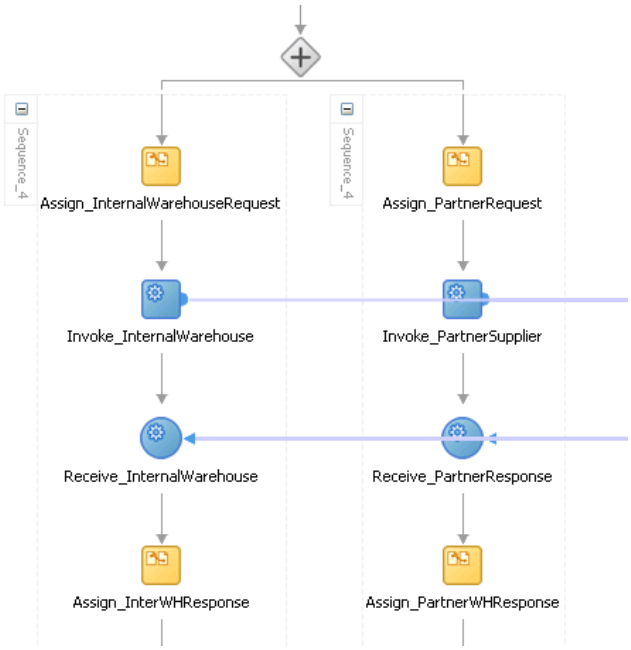
4. Drag and define additional activities onto each side of the flow to invoke multiple services at the same time. Figure 9–3 provides details.

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

```
<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>
```

BPEL Version 2.0 Example

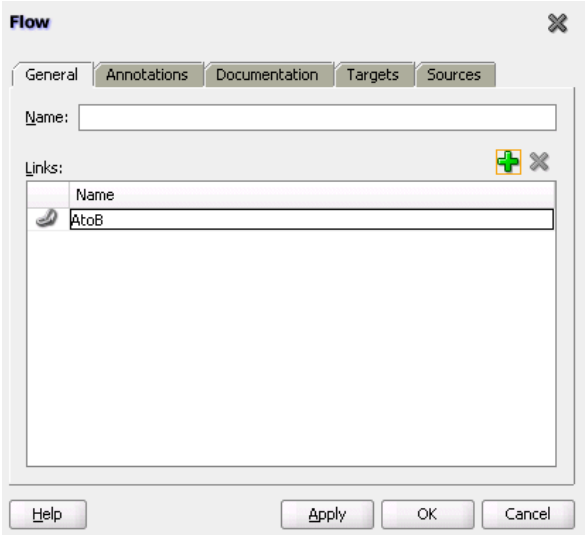
```
<flow>
      <links>
        <link name="AtoB"/>
      </links>
      <assign name="B">
        <targets>
          <target linkName="AtoB"/>
        </targets>
        <copy>
          <from>concat($output.payload,
            'B')</from>
          <to>$output.payload</to>
        </copy>
      </assign>
      <assign name="A">
        <sources>
          <source linkName="AtoB"/>
        </sources>
        <copy>
          <from>concat($output.payload,
           'A')</from>
          <to>$output.payload</to>
        </copy>
      </assign>
    </flow>
```

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.

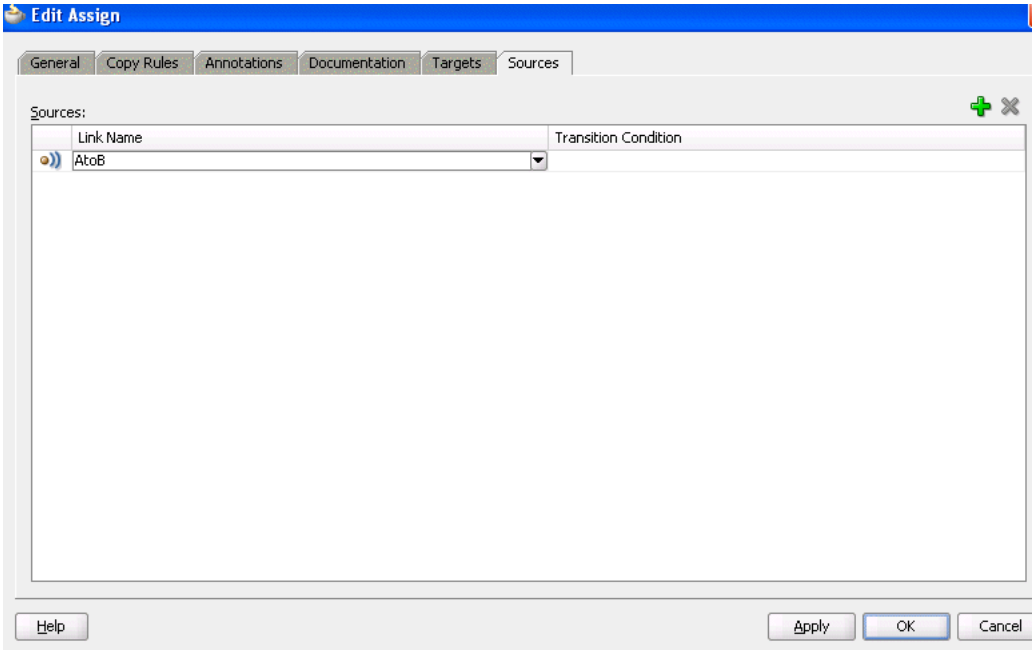
Figure 9–5 Link Name Creation



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

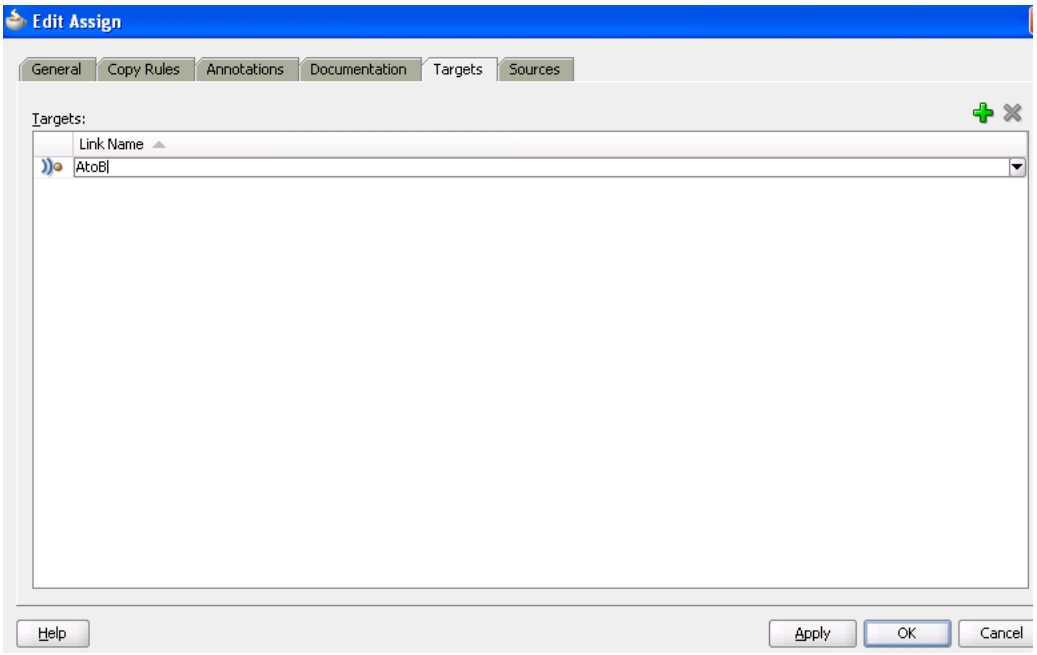


Note that each source activity can specify an optional **Transition Condition** as a safe guard for following the specified link. Click the row in this column to invoke the **Browser** icon for accessing the Expression Builder dialog for creating a condition. If the **Transition Condition** column is 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.

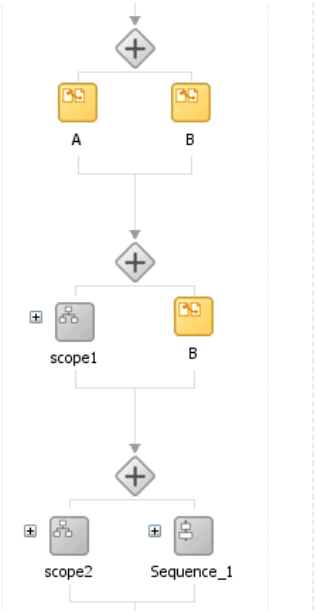
Figure 9–7 Target Activity



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>
        <copy>
          <from>concat($output.payload, 'A')</from>
          <to>$output.payload</to>
```

```
    </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.

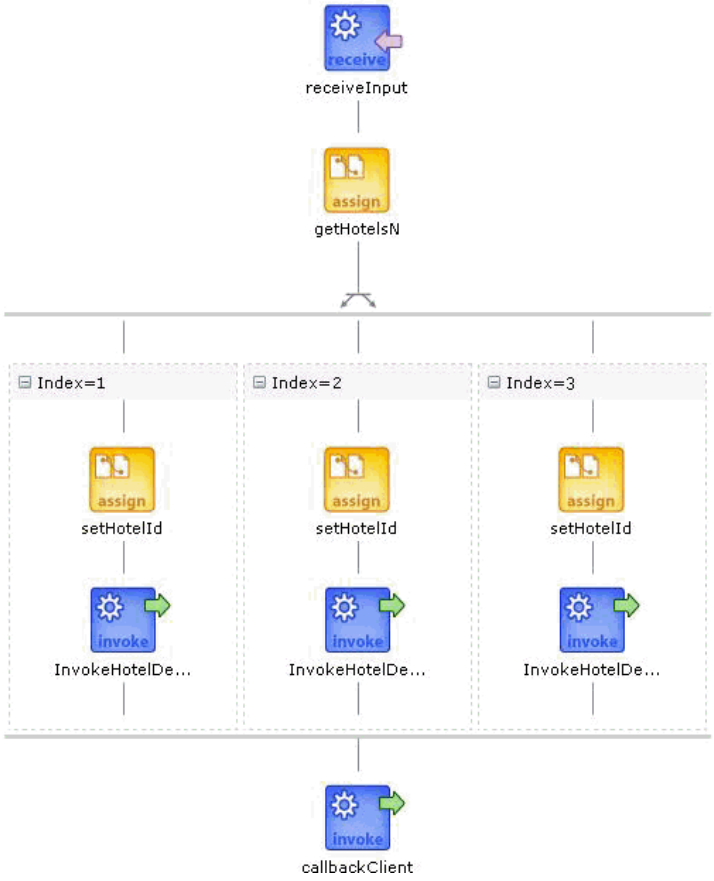
The branches created by flowN perform the same activities, but use different data. Each branch uses the index variable to look up input variables. The index variable can be used in the XPath expression to acquire the data specific for that branch.

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



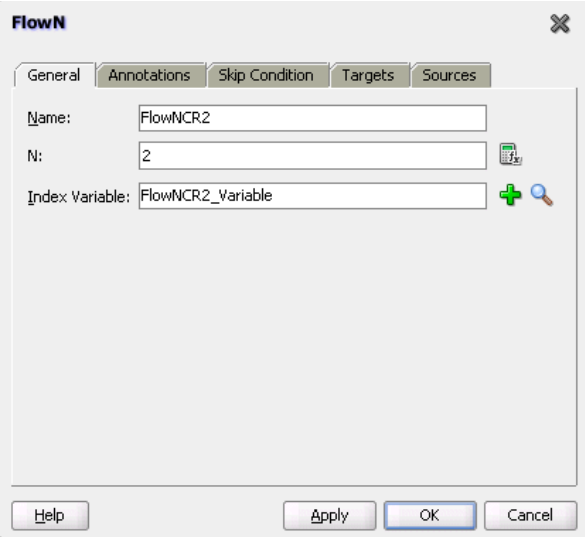
9.3.1.1 How to Create a flowN Activity

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 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
       noded passed in.
       An intermediate variable called "NbParallelFlow" is
       used to store the number of N flows being executed
       -->
    <assign name="getHotelsN">
      <copy>
        <from
expression="count($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.

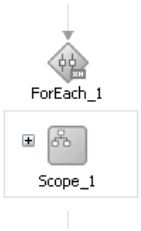
9.3.2.1 How to Create a forEach Activity

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.

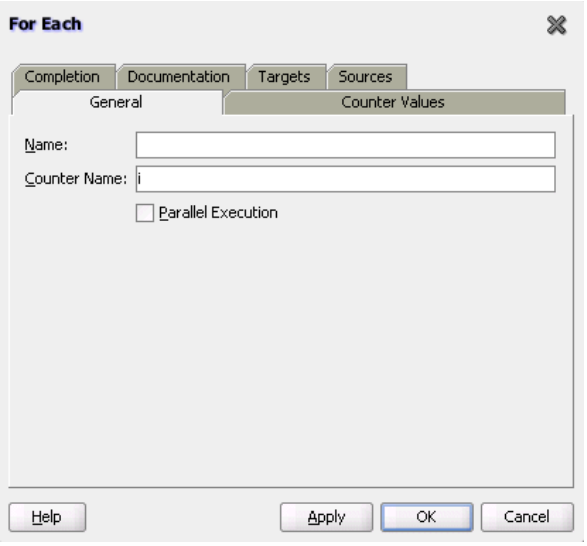
Figure 9–12 Contained Scope Activity in a forEach Activity



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.

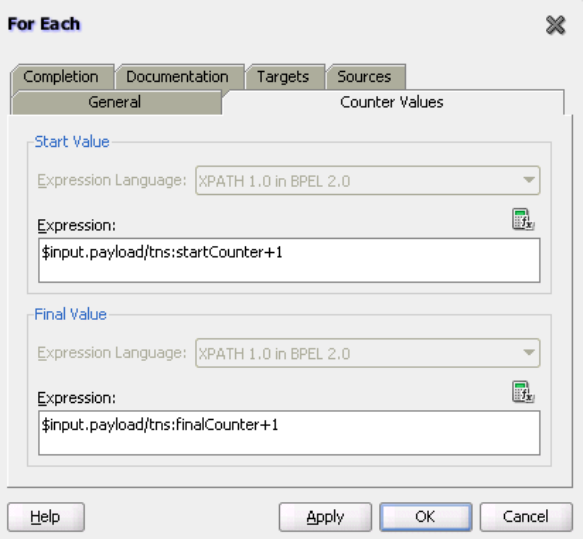
   Note the **Parallel Execution** checkbox. If this checkbox is selected, all looping iterations are started at the same time and processed in parallel.

Figure 9–13 General Tab of the forEach Activity



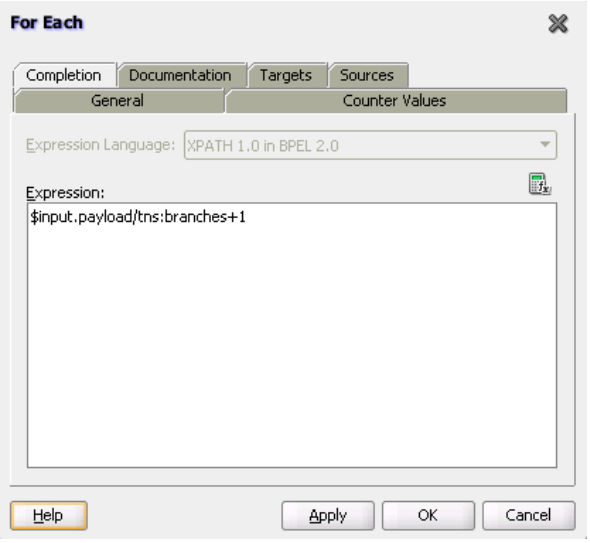
5. Click the **Counter Values** tab.
6. Enter the starting counter value and final counter value, as shown in Figure 9–14.

Figure 9–14 Counter Values Tab of the forEach Activity



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.

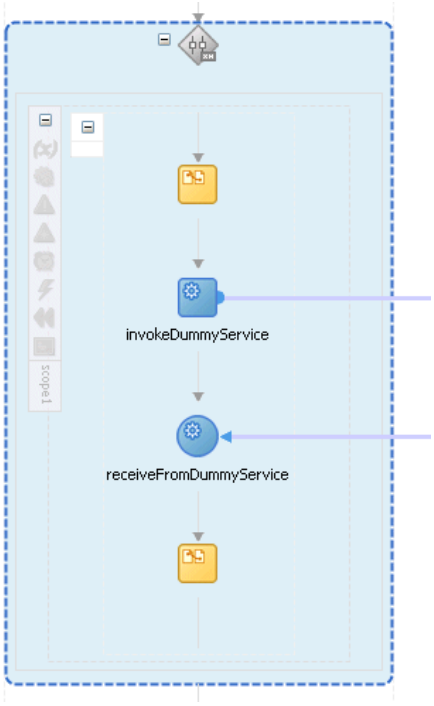
Figure 9–15 Completion Tab of the forEach Activity



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 `.bpel` 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>
  <!-- respond output to requestor -->
  <reply name="replyOutput" partnerLink="client"
      portType="tns:Test" operation="process" variable="output"/>
</sequence>
```

10

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:

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:

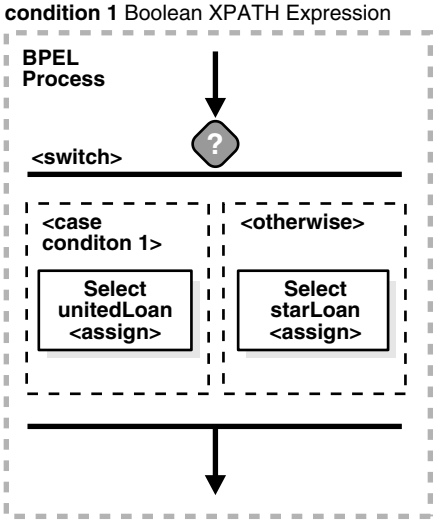
- Switch activity in a BPEL version 1.1 project
- If activity in a BPEL version 2.0 project

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



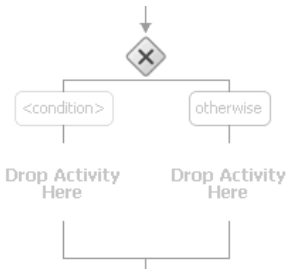
10.2.1.1 How to Create a Switch Activity

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:getVariableDate('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:

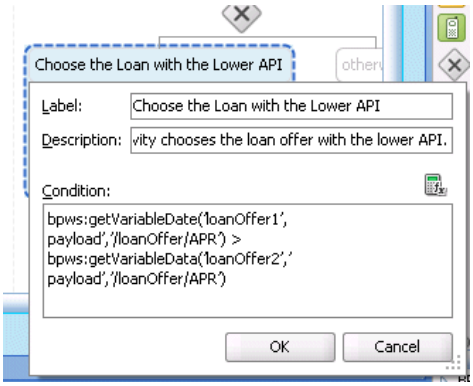
   - If `loanOffer1` has the higher APR, then the first branch selects `loanOffer2` by assigning the `loanOffer2` payload to the `selectedLoanOffer` payload.

- 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.

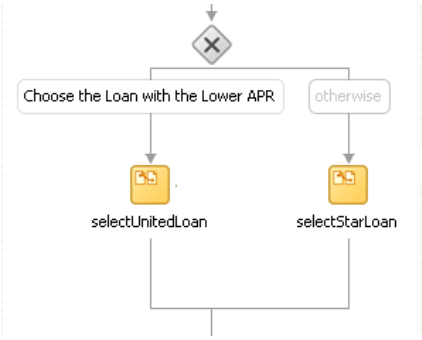
Figure 10–4 Completed Condition Dialog



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>
```

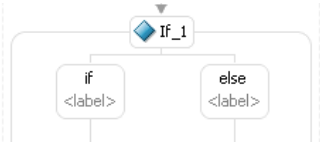
10.2.2.1 How to Create an If Activity

To create an If activity:

1. In the Component Palette, expand **BPEL Constructs**.
2. Drag an **If** activity into the designer.

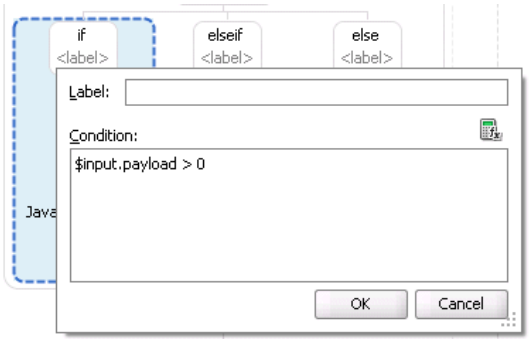
   The **if** and **else** conditions are displayed, as shown in Figure 10–6.

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



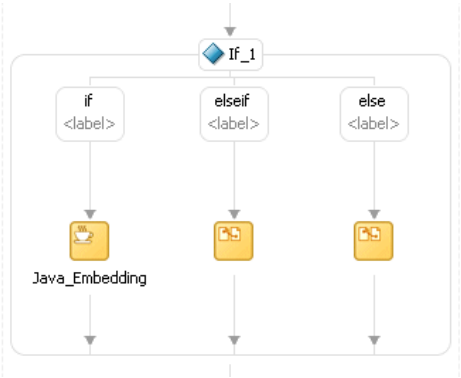
6. Click **OK**.
7. Drag and define additional activities into the **if** condition, as needed. These activities are executed if the if condition evaluates to true.
8. Click the **elseif** branch (if you added this branch).
9. In the **Condition** field, enter a condition, as shown in Figure 10–8.

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:

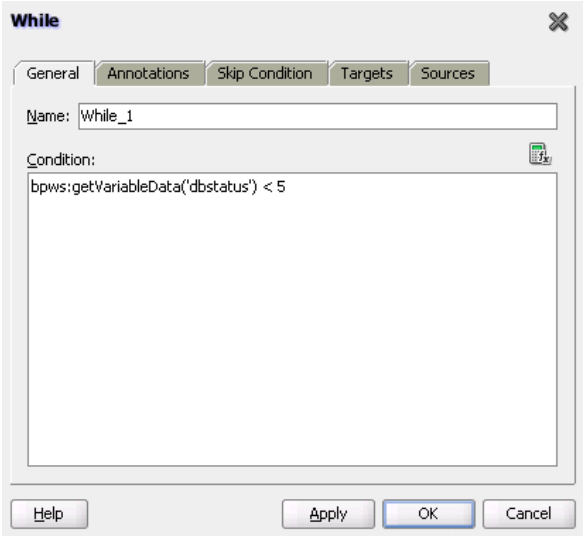
1. In the Component Palette, expand **BPEL Constructs**.

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

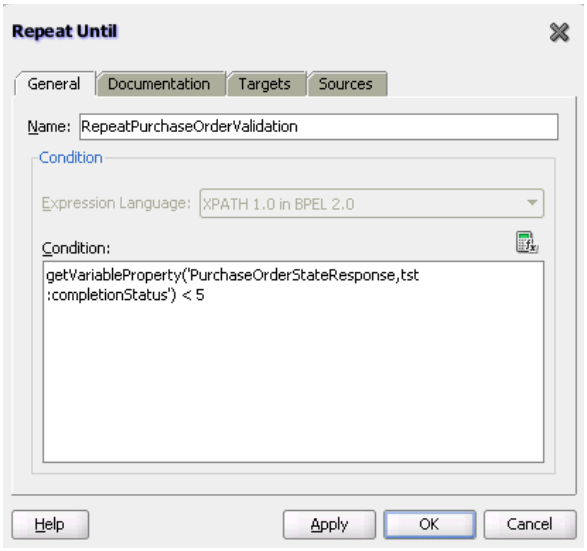
To create a repeatUntil activity:

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**.

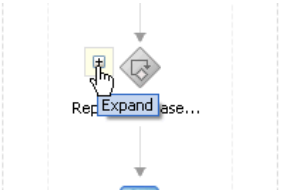
   The condition you entered is displayed in the Repeat Until dialog, as shown in Figure 10–11.

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.

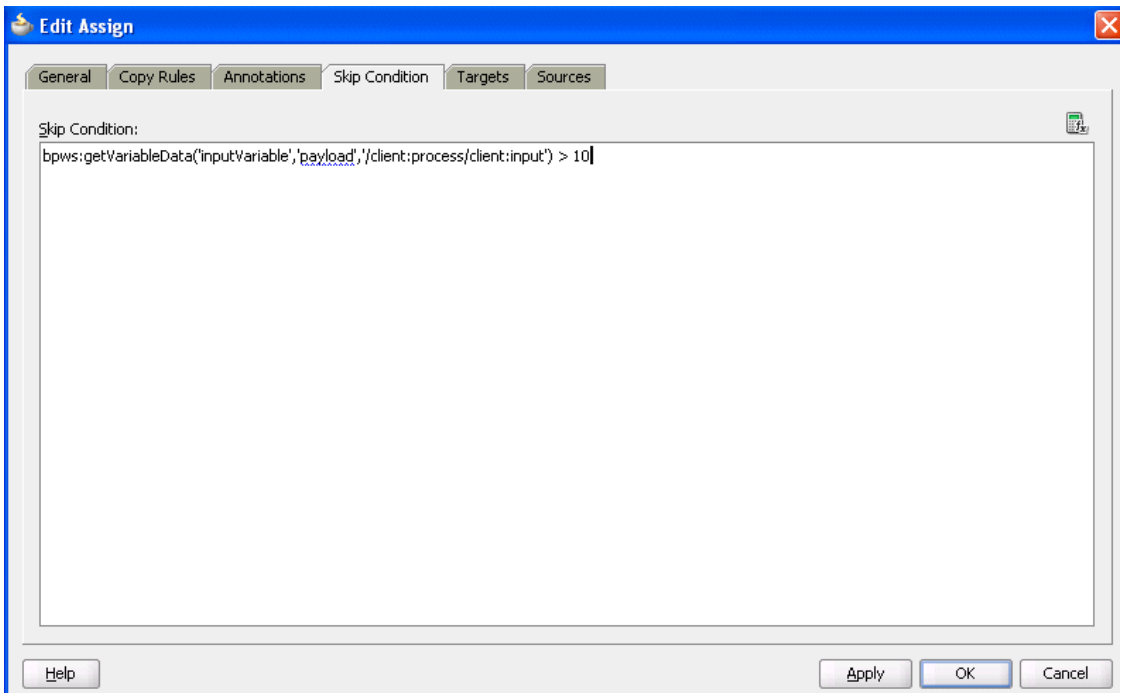
If the XPath engine fails to evaluate the boolean value, `bpws:subLanguageFault` is thrown. This is the same fault thrown when a switch/case condition does not evaluate to a boolean value. This is also logged to the audit trail for debugging purposes.

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','/clien
t:
 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>
```

11

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:

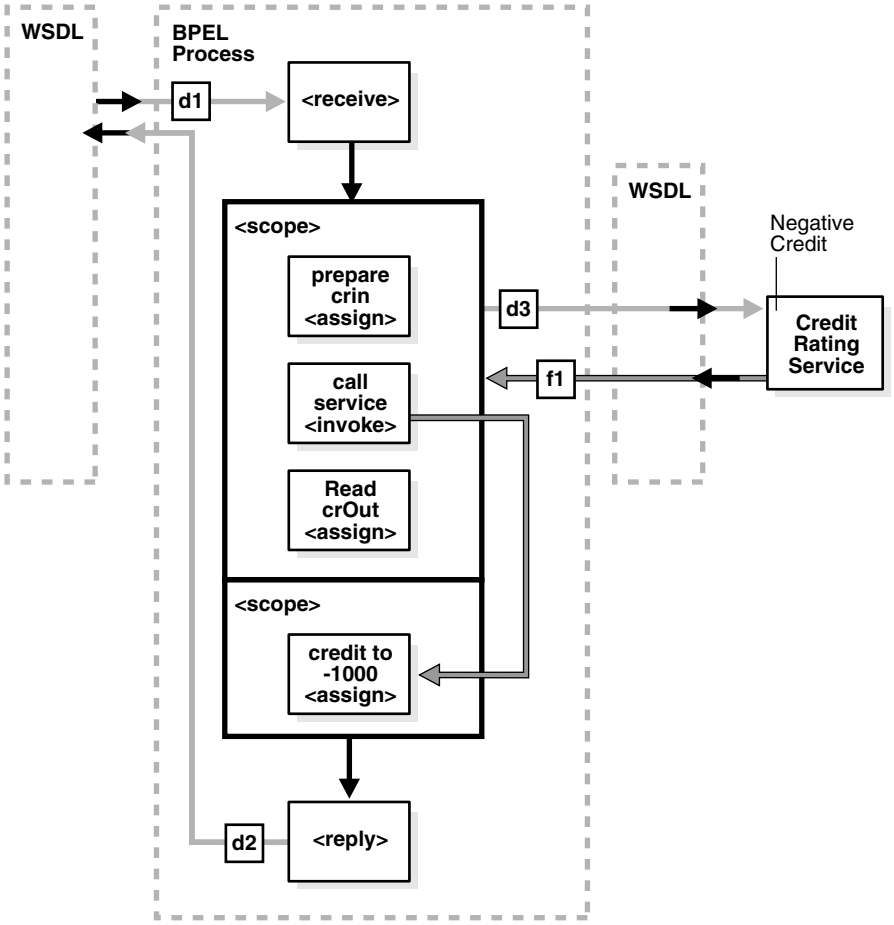
For additional information on creating fault handling in a SOA composite application, see the Fusion Order Demo application.

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>
```

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">oramds:/apps/faultpolicyfiles/
 fault-policies.xml
</property>
<property
 name="oracle.composite.faultBindingFile">oramds:/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">oramds:/apps/faultpolicyfiles/
 fault-policies.xml
</property>
<property
 name="oracle.composite.faultBindingFile">oramds:/apps/faultpolicyfiles/
 fault-bindings.xml
</property>
```

3. Define the `condition` section of the fault policy file.
   - Note the following details about the `condition` section:
     - This section provides a condition based on `faultName`.
     - Multiple conditions may be configured for a `faultName`.
     - Each condition has one `test` section (an XPath expression) and one `action` section.
     - The `test` section (XPath expression) is evaluated for the fault variable available in the fault.
     - The `action` section has a reference to the action defined in the same file.
     - You can only query the fault variable available in the fault.

– The order of condition evaluation is determined by the sequential order in the document.

Table 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"`
An `action` of `ora-terminate` is specified.
	`<condition>`
	`  <test>$fault.code="WSDLReading Error"`
	`  </test>`
	`  <action ref="ora-terminate"/>`
	`</condition>`

No `test` condition is provided. This is a catchAll condition for a given `faultName`.	`<condition>`
	`  <action ref="ora-rethrow"/>`
	`</condition>`

If the `faultName` name attribute is missing, this indicates a catchAll activity for faults that have any `QName`.	`<faultName > . . . </faultName>`

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 *N* 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 *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.
	`<Action id="ora-retry">`
	`  <Retry>`
	`  <retryCount>3</retryCount>`
	`  <retryInterval>2</retryInterval>`
	`  <exponentialBackoff/>`
	`  <retryFailureAction ref="ora-java"/>`
	`  <retrySuccessAction ref="ora-java"/>`
	`  </Retry>`
	`</Action>`
	
	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.	`<Action id="ora-human-intervention">` ` <humanIntervention/></Action>`
Terminate Process: Terminates the process	`<Action id="ora-terminate"><abort/></Action>`

Java Code: Enables you to execute an external Java class.
`returnValue`: 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."
	`<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="RETRHOW"`
	`  ref="ora-rethrow-fault"/>`
	`  <returnValue value="ABORT"`
	`  ref="ora-terminate"/>`
	`  <returnValue value="RETRY" ref="ora-retry"/>`
	`  <returnValue value="MANUAL"`
	`  ref="ora-human-intervention"/>`
	`</javaAction>`
	`</Action>`

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.	`<Action id="ora-rethrow-fault"><rethrowFault/></Action>`
Replay Scope: Raises a replay fault.	`<Action id="ora-replay-scope"><replayScope/></Action>`

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>
        <Action id="ora-terminate">
            <abort/>
        </Action>
        <Action id="ora-java">
            <!-- this is user provided class-->
            <javaAction
 className="com.oracle.bpel.client.config.faultpolicy.TestJavaAction"
 defaultAction="ora-terminate" propertySet="prop-for-billing">
                <returnValue value="REPLAY" ref="ora-terminate"/>
                <returnValue value="RETRHOW" ref="ora-rethrow-fault"/>
                <returnValue value="ABORT" ref="ora-terminate"/>
                <returnValue value="RETRY" ref="ora-retry"/>
                <returnValue value="MANUAL" ref="ora-human-intervention"/>
            </javaAction>
        </Action>

                    </Actions>
                  <Properties>
                          <propertySet name="prop-for-billing">
            <property name="user_email_recipient">bpeladmin</property>
            <property name="email_recipient">joe@abc.com</property>
            <property name="email_recipient">mike@xyz.com</property>
            <property name="email_threshold">10</property>
            <property name="sms_recipient">+429876547</property>
            <property name="sms_recipient">+4212345</property>
            <property name="sms_threshold">20</property>
            <property name="user_email_recipient">john</property>
        </propertySet>
        <propertySet name="prop-for-order">
            <property name="email_recipient">john@abc.com</property>
            <property name="email_recipient">jill@xyz.com</property>
            <property name="email_threshold">10</property>
            <property name="sms_recipient">+42222</property>
            <property name="sms_recipient">+423335</property>
            <property name="sms_threshold">20</property>
        </propertySet>

                  </Properties>
</faultPolicy>
<faultPolicy version="2.0.1"
                  id="Billing_ServiceFaults"
                  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"

                  xmlns="http://schemas.oracle.com/bpel/faultpolicy"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<Conditions>
    <faultName>
    <condition>
       <action ref="ora-manual"/>
    </condition>
    </faultName>
</Conditions>
<Actions>
        <Action id="ora-manual">
            <humanIntervention/>
        </Action>
</Actions>
</faultPolicy>
```

```
</faultPolicies>
```

Example 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_SeriveFaults -->
    <component faultPolicy="CRM_ServiceFaults">
        <name>HelloWorld</name>
        <name>ShippingComponent</name>
        <name>AnotherComponent"</name>
    </component>
    <!-- Below listed reference names and port types use polic CRM_ServiceFaults
 -->
    <reference faultPolicy="CRM_ServiceFaults">
        <name>creditRatingService</name>
        <name>anotherReference</name>
        <portType
 xmlns:credit="http://services.otn.com">credit:CreditRatingService</portType>
        <portType
 xmlns:db="http://xmlns.oracle.com/pcbpel/adapter/db/insert/">db:insert_
plt</portType>
    </reference>
    <reference faultPolicy="test1">
        <name>CreditRating3</name>
    </reference>
</faultPolicyBindings>
```

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="RETRHOW" 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
`<ReturnValue value="" ref="ora-rethrow"/>`	Fails in validation.
`<javaAction className="mypackage.myclass" defaultAction="ora-human-intervention ">`	Execute `ora-human-intervention` after Java code execution. This attribute is used if the return from the method does not match any provided `ReturnValue`.
`<ReturnValue value="RETRY" ref="ora-retry"/> <ReturnValue value="" ref=""/>`	Fails in validation.
`<javaAction className="mypackage.myclass" defaultAction="ora-human-intervention"> <ReturnValue></ReturnValue>`	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 Correleation 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

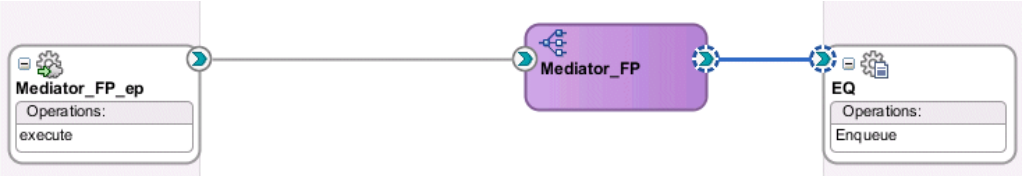
The fault policy retry action may not execute with multiple faults in the same flow. This may be because the retry count has already been reached for any of the previous faults.

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

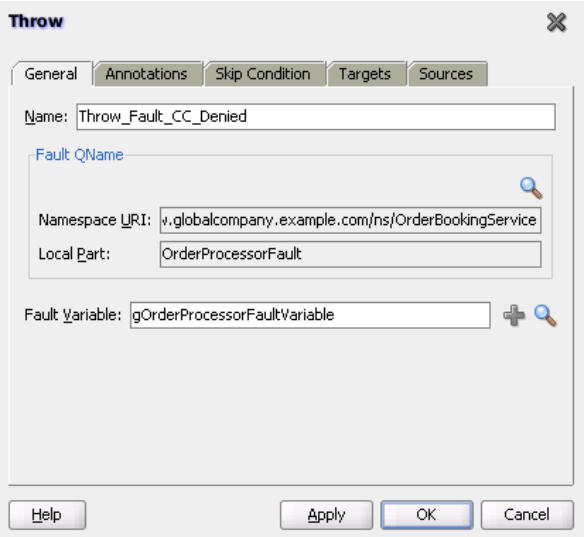
To create a throw activity:

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



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.

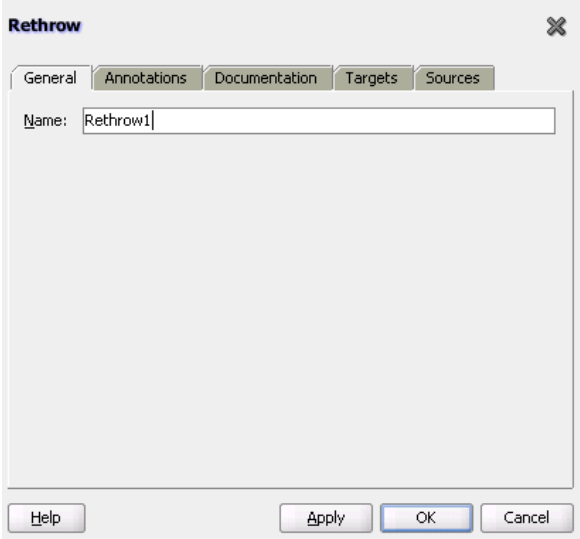
11.8.1 How to Create a Rethrow Activity

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.

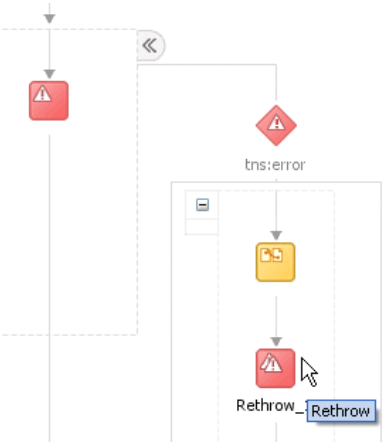
Figure 11–4 Rethrow Dialog



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 `.bpel` 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 partnerlinke="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

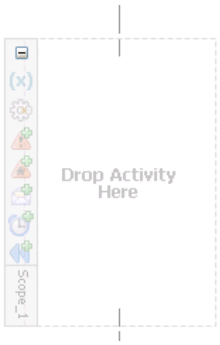
```
<scope name="Scope_FulfillOrder">
    <variables>
        <variable name="lFulfillOrder_InputVariable"
        messageType="ns17:requestMessage"/>
    </variables>
    <sequence>
        <assign name="Assign_OrderData">
            <copy>
                <from variable="gOrderInfoVariable"
                    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.

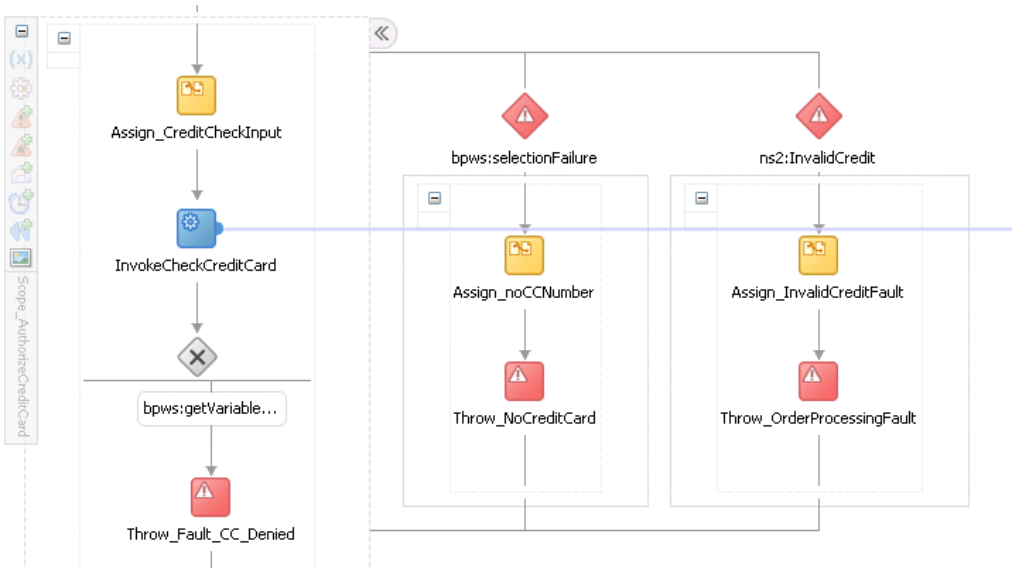
Figure 11–6 Expanded Scope Activity



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:

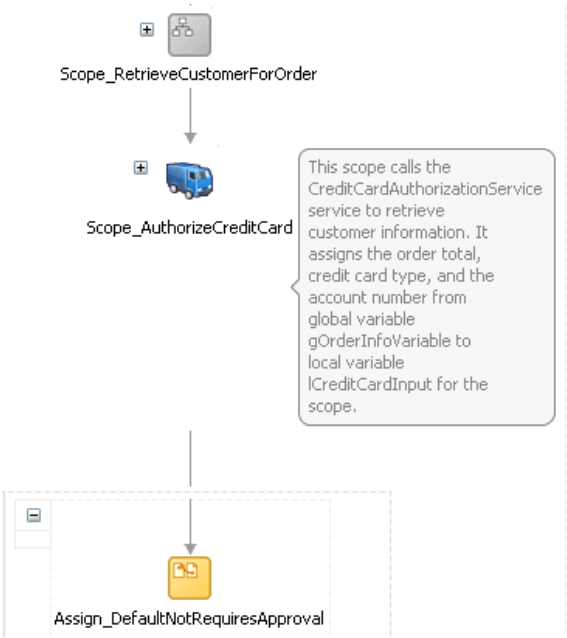
1. Perform one of the following steps:
   - Right-click the scope and select **User Documentation**.
   - Double-click the scope and select the **User Documentation** tab.

   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>
<faultHandlers>
    <catch faultName="bpws:selectionFailure">
        <sequence>
             <assign name="Assign_noCCNumber">
                 <copy>
                     <from expression="string('CreditCardCheck - NO
                         CreditCard')"/>
                     <to variable="gOrderProcessorFaultVariable"
                         part="code"/>
                 </copy>
             </assign>
             <throw name ="Throw_NoCreditCard"
                 faultVariable="gOrderProcessorFaultVariable"
                 faultName="ns9:OrderProcessingFault"/>
        </sequence>
    </catch>
    <catch faultName="ns2:InvalidCredit">
        <sequence>
            <assign name="Assign_InvalidCreditFault">
                <copy>
                    <from expression="concat(bpws:getVariableData
                         ('gOrderInfoVariable','/ns4:orderInfoVOSDO/
                         ns4:CardTypeCode'), ' is not a valid
                         creditcard type')"/>
                    <to variable="gOrderProcessorFaultVariable"
                        part="summary"/>
                </copy>
                <copy>
                    <from expression="string('CreditCardCheck - NOT VALID')"/>
                    <to variable="gOrderProcessorFaultVariable"
                        part="code"/>
                </copy>
            </assign>
            <throw name="Throw_OrderProcessingFault"
                faultName="ns9:OrderProcessingFault"
                faultVariable="gOrderProcessorFaultVariable"/>
        </sequence>
    </catch>
</faultHandlers>
<sequence>
    <assign name="Assign_CreditCheckInput">
        <copy>
            <from variable="gOrderInfoVariable"
                query="/ns4:orderInfoVOSDO/ns4:OrderTotal"/>
            <to variable="lCreditCardInput" part="Authorization"
                query="/ns8:AuthInformation/ns8:PurchaseAmount"/>
        </copy>
        <copy>
            <from variable="gOrderInfoVariable"
                query="/ns4:orderInfoVOSDO/ns4:CardTypeCode"/>
                    <to variable="lCreditCardInput" part="Authorization"
                         query="/ns8:AuthInformation/ns8:CCType"/>
        </copy>
        <copy>
            <from variable="gOrderInfoVariable"
                query="/ns4:orderInfoVOSDO/ns4:AccountNumber"/>
            <to variable="lCreditCardInput" part="Authorization"
                query="/ns8:AuthInformation/ns8:CCNumber"/>
                </copy>
```

```
        </assign>
        <invoke name="InvokeCheckCreditCard"
            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>
```

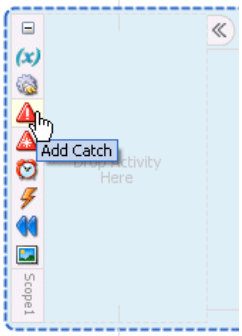
```
    <catchAll>
          <empty/>
       </catchAll>
</faulthandlers>
```

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



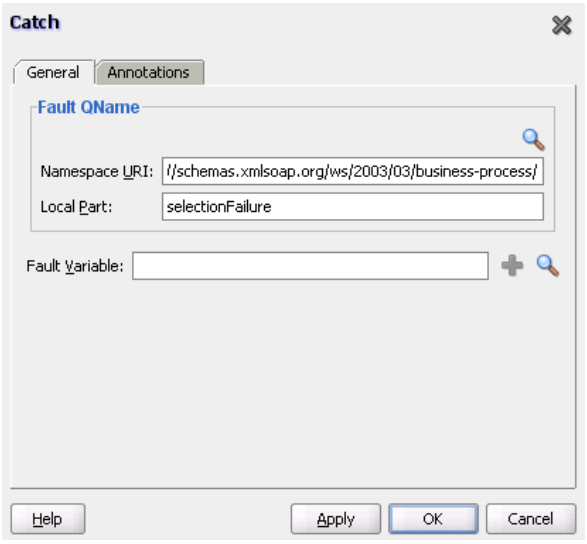
   This creates a catch activity in the right side of the scope activity.

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.

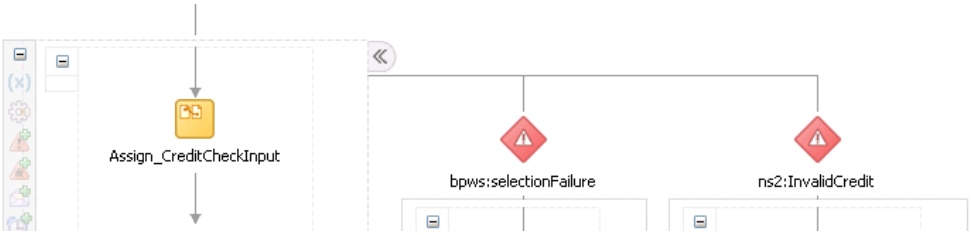
Figure 11–10 Catch Dialog



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

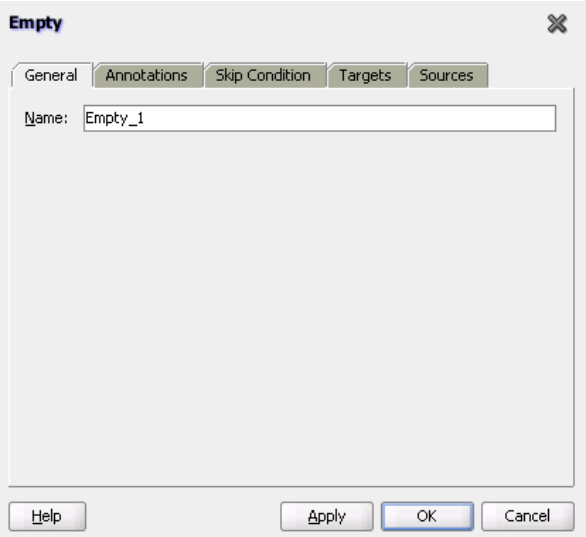
There is often a need to use an activity that does nothing. An example is when a fault must be caught and suppressed. In this case, you can use the empty activity to insert a no-op instruction into a business process.

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

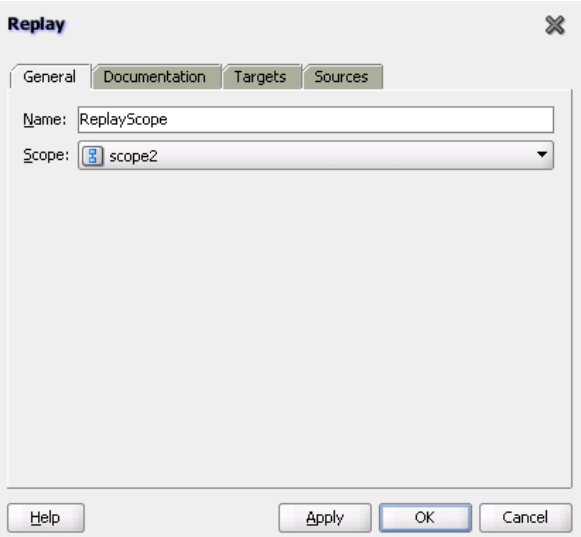
You can create a replay activity inside a scope activity to re-execute all of the activities inside the scope.

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.

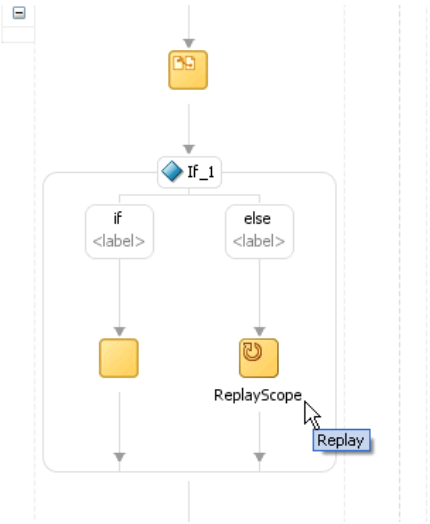
Figure 11–13 Replay Dialog



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.

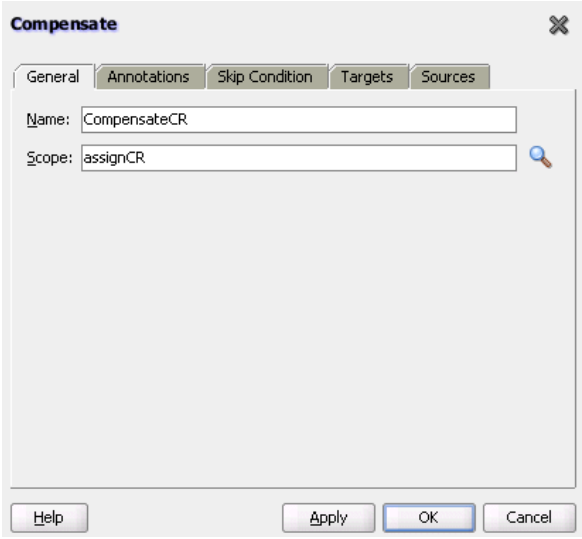
The compensate form, in which the scope name is omitted in a compensate activity, explicitly invokes this default behavior. This is useful when an enclosing fault or compensation handler must perform additional work, such as updating variables or sending external notifications, in addition to performing 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



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

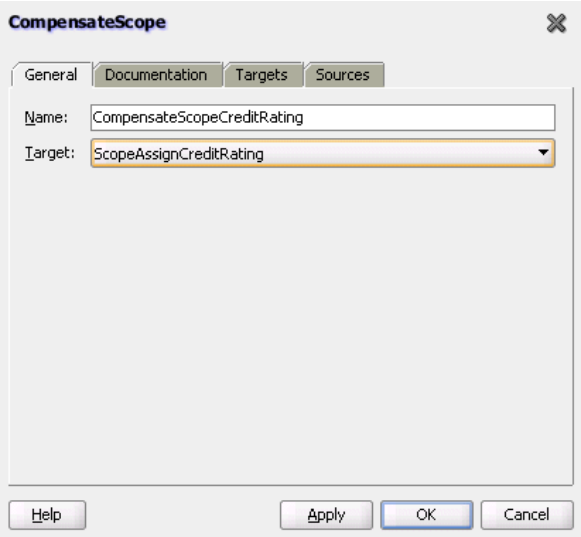
Note: This activity is supported in BPEL 2.0 projects.

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.

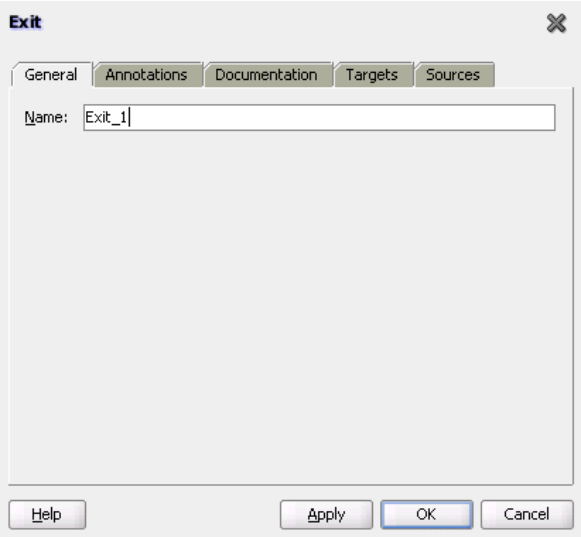
Note: Any open conversations are also impacted by the exit activity. For example, other partners interacting with the process may wait for a response that never arrives.

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.

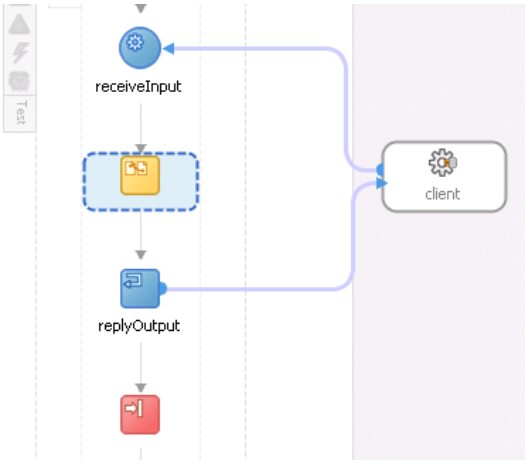
Figure 11–18 Exit Activity



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 `.bpel` 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 `services:NegativeCredit` fault is thrown.
Greater than or equal to zero	The assertion is correct and the second assertion is evaluated.
Less than 600	The `services:InsufficientCredit` 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

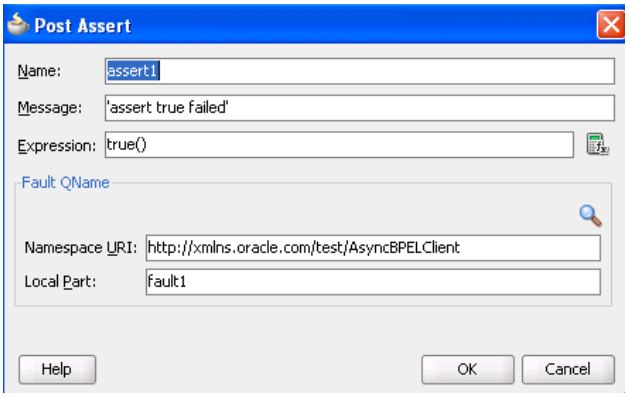
- In standalone assert activities

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.

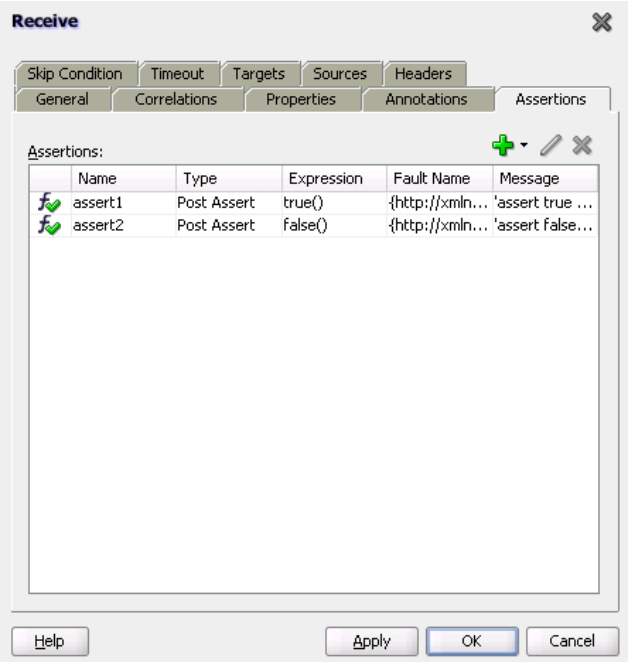
   Based on your selection, the Pre Assert or Post Assert dialog is displayed.
8. Specify values for the assertion condition, as shown in Figure 11–20. For this example, Post Assert was selected for an assertion condition on a receive activity.
   a. Select the **Fault QName** to be thrown by clicking the **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



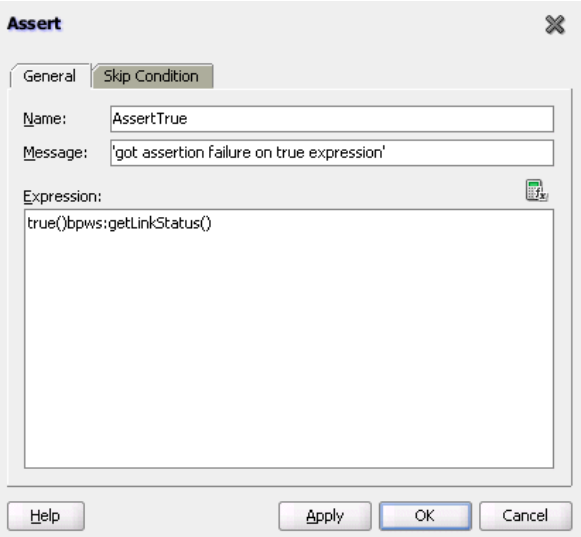
10. Click **Apply**, then **OK**.

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="$crOutput.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'"
```

12

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:

12.1 Introduction to Transaction Semantics

Transaction semantics in release 11*g* 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 11*g*, 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 bpel.config.transaction Set to required...	With bpel.config.transaction Set to requiresNew...
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 `bpel.config.oneWayDeliveryPolicy` is set to `sync`.	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 `<reply>`).	Is saved.	Gets the fault and can catch it.
Throws a fault that is not handled (that is, it uses `<throw>`).	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 `bpelx:rollback` fault (that is, it uses `<throw>`).	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 `<reply>`).	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 `<throw>`).	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 `<throw>`) a `bpelx:rollback` 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 bpelx:rollback...
`onewayDeliveryPolicy=async.persist` (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 bpelx:rollback...
`onewayDeliveryPolicy=sync` and `transaction=requiresNew` (The BPELCallee runs in the same thread, but a different transaction.)	The BPELCaller receives a `FabricInvocationException`. The BPELCallee transaction rolls back if the fault is not handled.	The BPELCaller receives a `FabricInvocationException`. 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 `FabricInvocationException`. The BPELCaller has a chance to handle the fault.	The whole transaction is rolled back.

13

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:

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));
```

```
        } 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>
```

```
    <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
`Object lookup( String name )`	JNDI access
`long getInstanceId( )`	Unique ID associated with each instance
`String setTitle( String title ) / String getTitle()`	Title of this instance
`String setStatus( String status ) / String getStatus()`	Status of this instance
`void setCompositeInstanceTitle(String title)`	Set the composite instance title

Table 13–1 (Cont.) Built in Methods for bpelx:exec

Method Name	Description
`void setIndex( int i, String value ) / String getIndex( int i )`	Six indexes can be used for a search
`void setCreator( String creator ) / String getCreator()`	Who initiated this instance
`void setCustomKey( String customKey ) / String getCustomKey()`	Second primary key
`void setMetadata( String metadata ) / String getMetadata ()`	Metadata for generating lists
`String getPreference( String key )`	Access preference
`void addAuditTrailEntry(String message, Object detail)`	Add an entry to the audit trail
`void addAuditTrailEntry(Throwable t)`	Access file stored in the suitcase
`Object getVariableData(String name) throws BPELFault`	Access and update variables stored in the scope
`Object getVariableData(String name, String partOrQuery) throws BPELFault`	Access and update variables
`Object getVariableData(String name, String part, String query)`	Access and update variables
`void setVariableData(String name, Object value)`	Set variable data
`void setVariableData(String name, String part, Object value)`	Set variable data
`void setVariableData(String name, String part, String query, Object value)`	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.

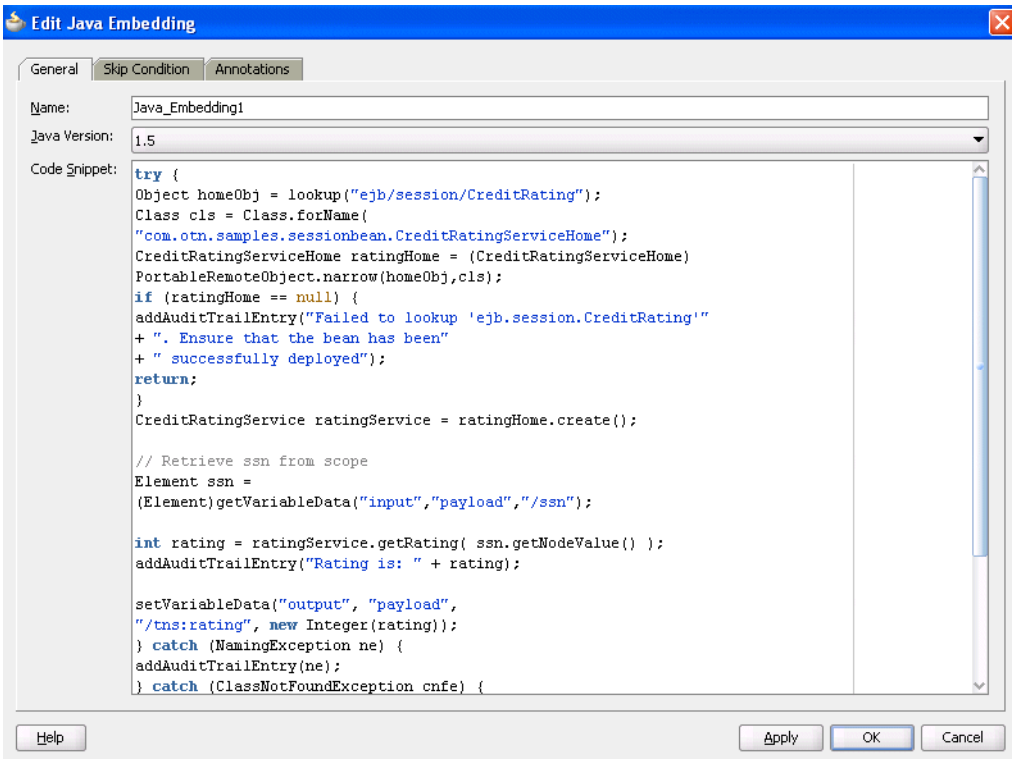
> **Note:** For custom classes, you must include any JAR files required for embedded Java code in the **BpelcClasspath** property in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control. See Section 13.3.1, "How to Add Custom Classes and JAR Files" for instructions. The JAR 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.

14

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:

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

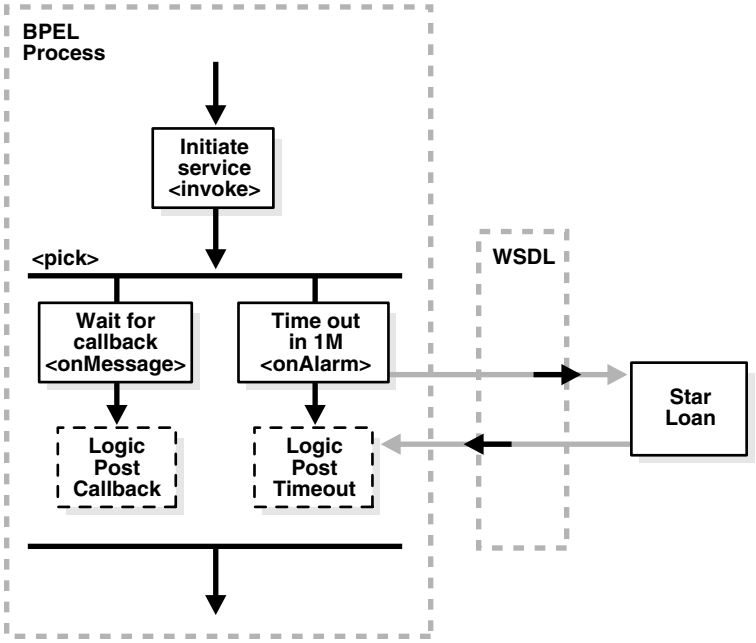
     This condition has code for a timeout of one minute. This time is defined as `PT1M`, which means to wait one minute before timing out. In this timeout setting:
     - `S` stands for seconds
     - `M` for one minute
     - `H` for hour
     - `D` for day
     - `Y` for year

     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:

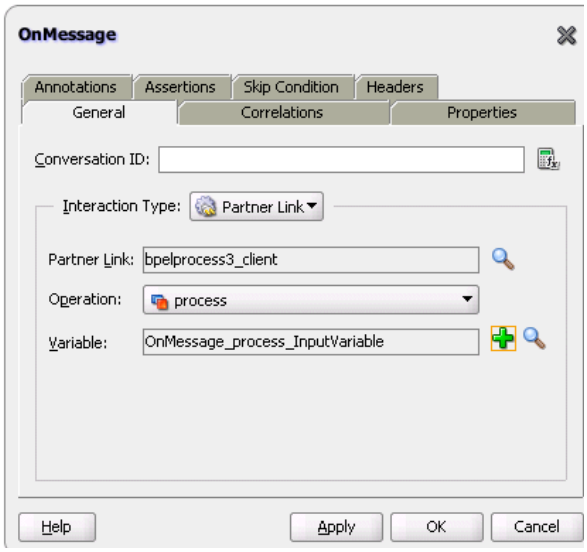
1. In the SOA Composite Editor, double-click the BPEL process service component.
2. In the Component Palette, expand **BPEL Constructs**.
3. Drag a **Pick** activity into the designer.

   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.

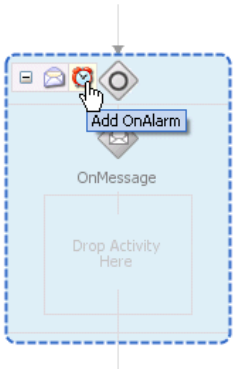
Figure 14–3 onMessage Branch



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.

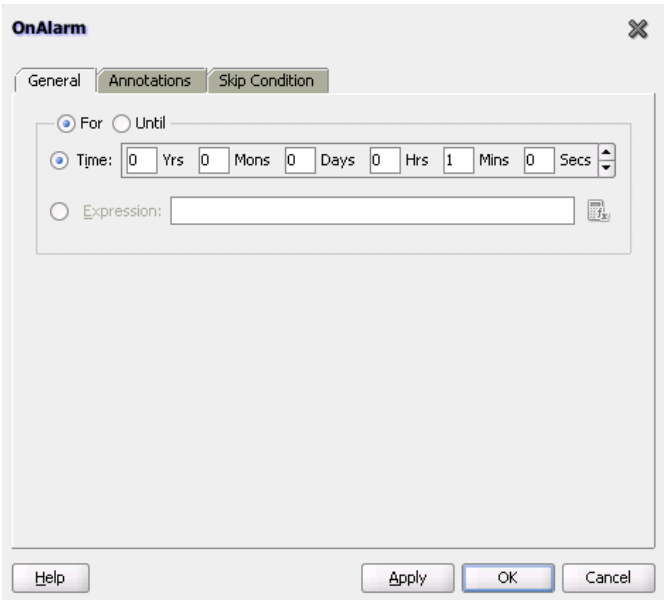
Figure 14–4 onAlarm Branch Creation



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 timesout -->
      <onAlarm for="PT1M">
           <assign>
               <copy>
                   <from>
                       <loanOffer xmlns="http://www.autoloan.com/ns/autoloan">
                           <providerName>Expired</providerName>
                           <selected type="boolean">false</selected>
                           <approved type="boolean">false</approved>
                           <APR type="double">0.0</APR>
                       </loanOffer>
                   </from>
                   <to variable="loanOffer" part="payload"/>
               </copy>
           </assign>
      </onAlarm>
</pick>
```

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 11*g* Release 1, instances are locked optimistically (as opposed to pessimistic locking in Release 10*g*). 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.

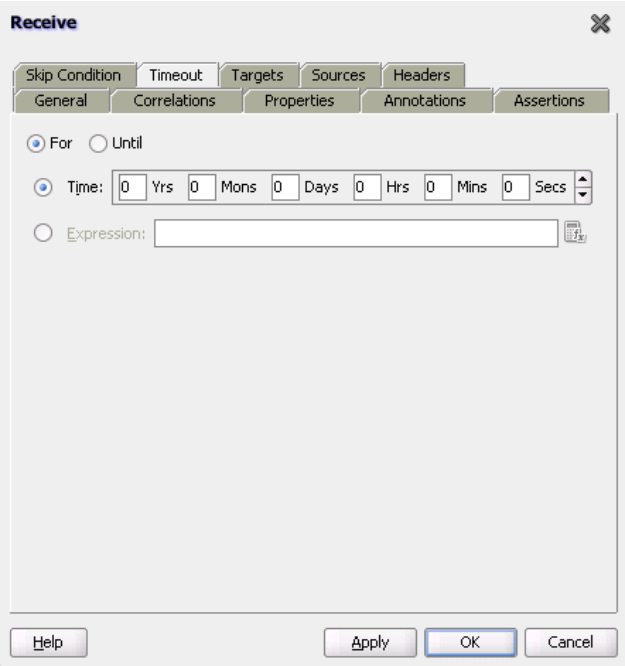
14.3.7 How to Set Timeouts for Request-Response Operations in Receive Activities

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



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**.

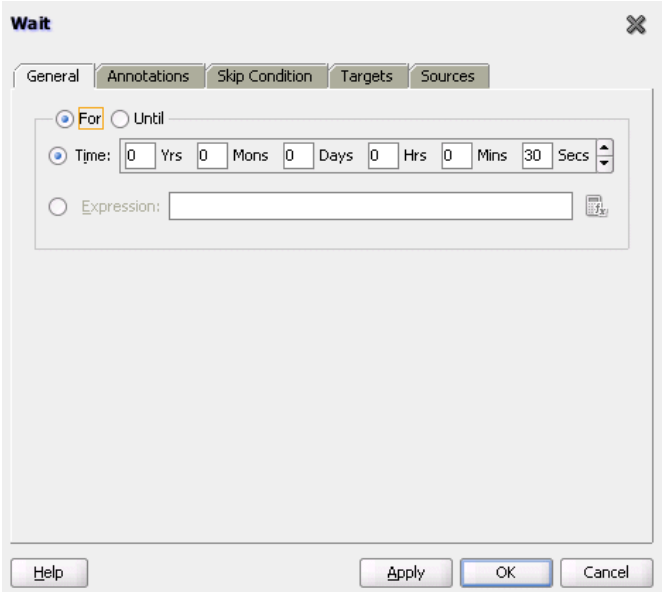
14.4.2 How to Create a Wait Activity

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

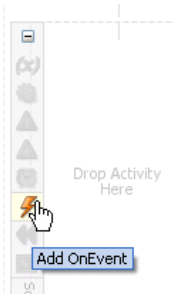
ends. When enabled, any number of events can occur. They are processed in parallel to the scope's primary activity and in parallel to each other. Message events also represent services operations exposed by a process and modeled as onEvent elements. Event handlers cannot create new process instances. 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

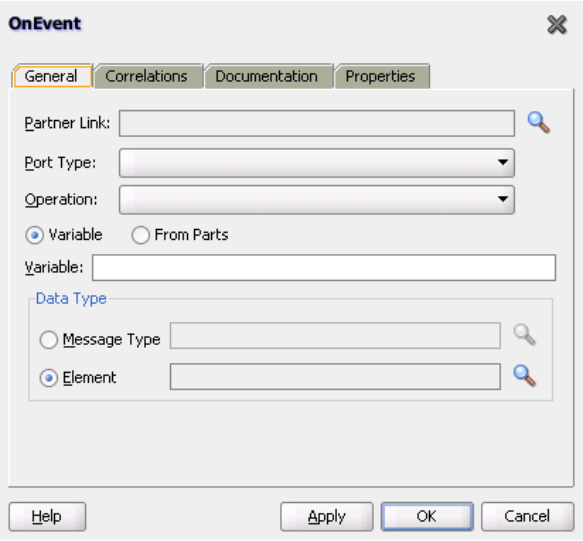


   This creates an **OnEvent** branch and an enclosed scope activity.

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."

15

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:

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.

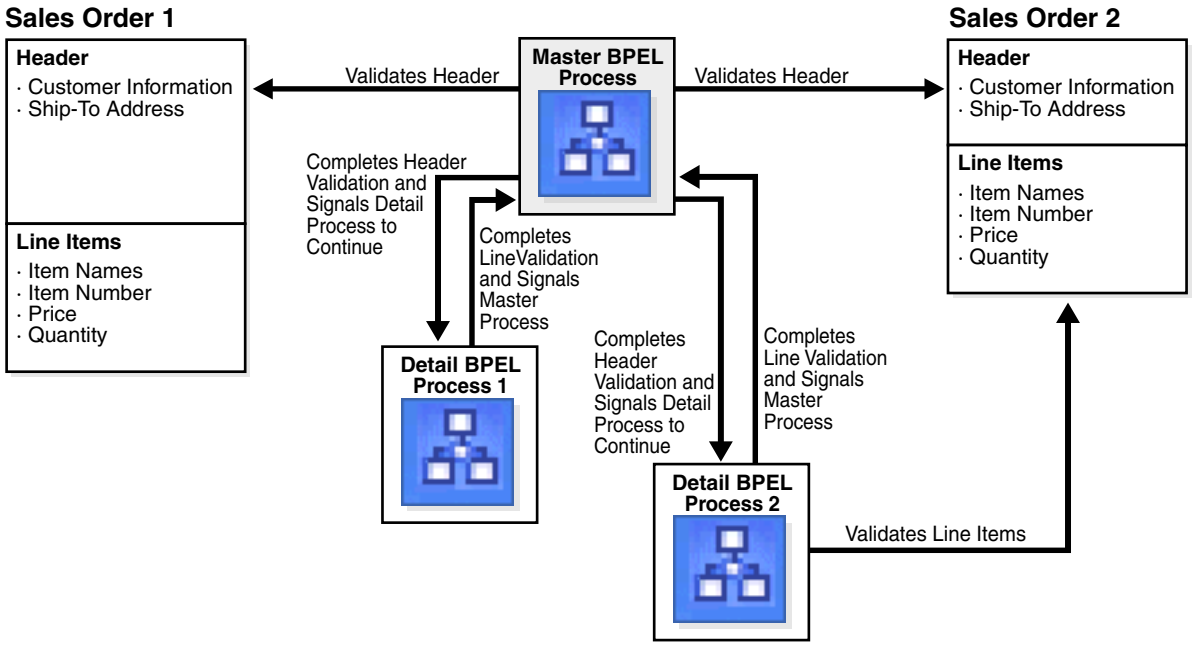
Potential coordination points are as follows:

- 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)



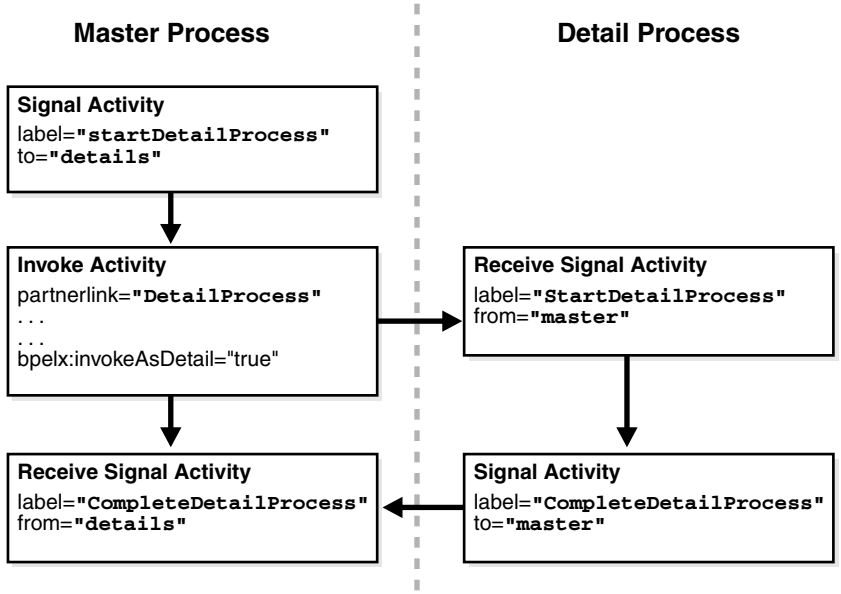
The following BPEL process activities coordinate actions between the master and detail processes:

- 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"/>
```

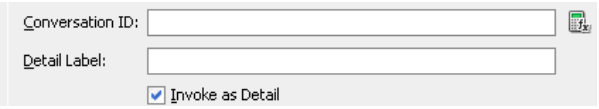
Note: In BPEL 2.0, the signal activity syntax is slightly different. The signal activity is wrapped in an `extensionActivity` element.

```
<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



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

<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.

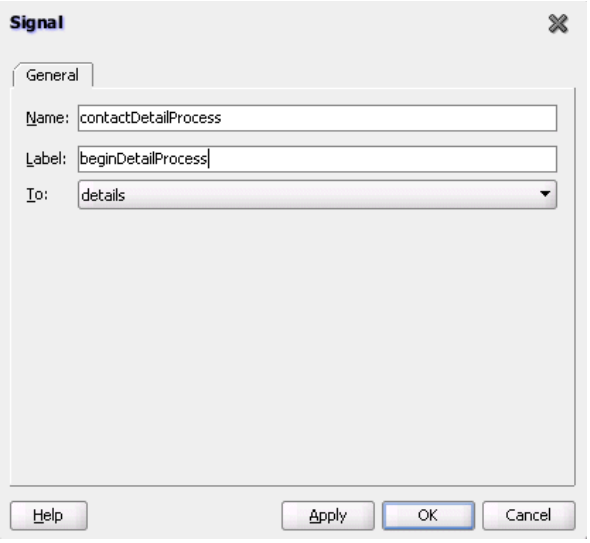
   This activity signals the detail process to perform processing at runtime.
6. Enter the details described in Table 15–2:

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.

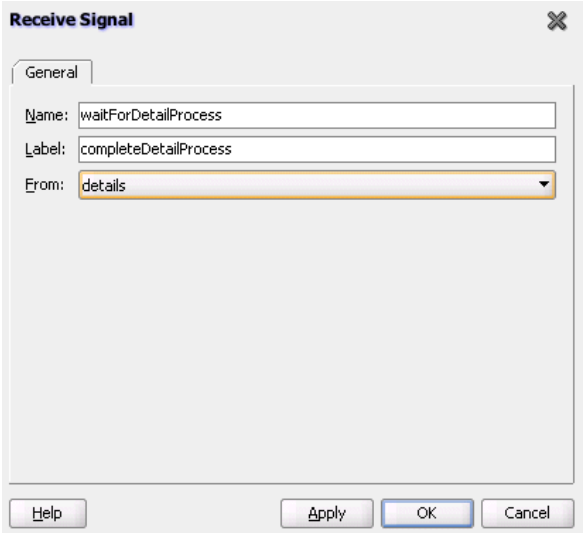
   This activity enables the master process to wait until it receives the signal executed by all of its detail processes.
10. Enter the details shown in Table 15–3:

Table 15–3 Receive Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, `waitForDetailProcess`).
Label	Enter a label name (for this example, `completeDetailProcess`). This label must match the signal activity label you set in the detail process in Step 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



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.

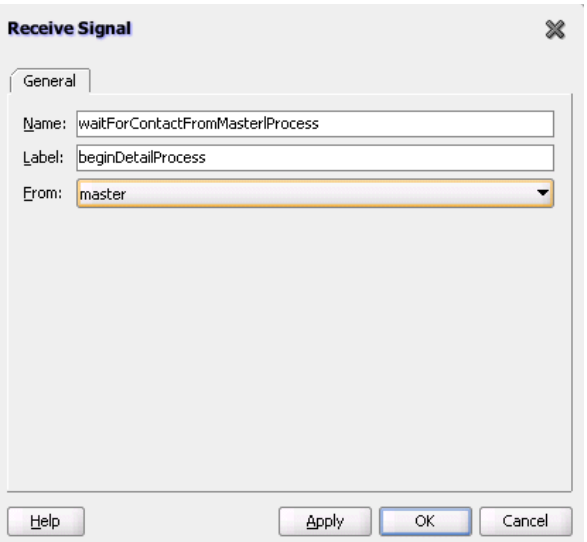
   This activity enables the detail process to wait until it receives the signal executed by its master process.
6. Enter the details shown in Table 15–4:

Table 15–4 Receive Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, `WaitForContactFromMasterProcess`).
Label	Enter a label name (for this example, `beginDetailProcess`). This label must match the signal activity label you set in the master process in Step 6 on page 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.

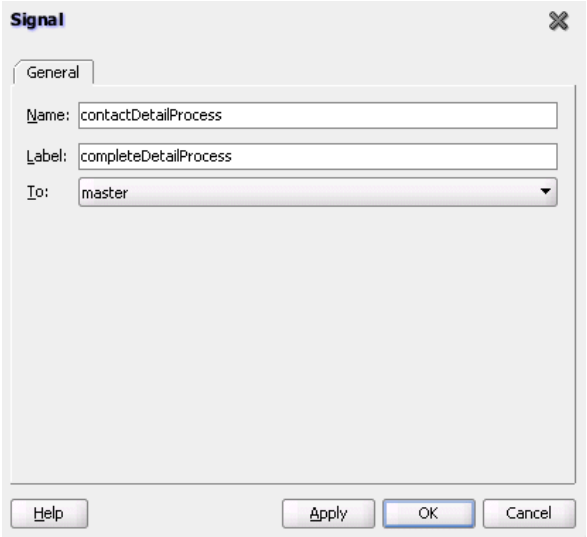
   This activity enables the detail process to signal its associated master process at runtime that processing is complete.
10. Enter the details described in Table 15–5:

Table 15–5 Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, `contactMasterProcess`).
Label	Enter a label name (for this example, `completeDetailProcess`). This label must match the receive signal activity label you set in the master process in Step 10 on page 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.

16

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:

Note: The fax and pager notification channels are not supported in 11*g* 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

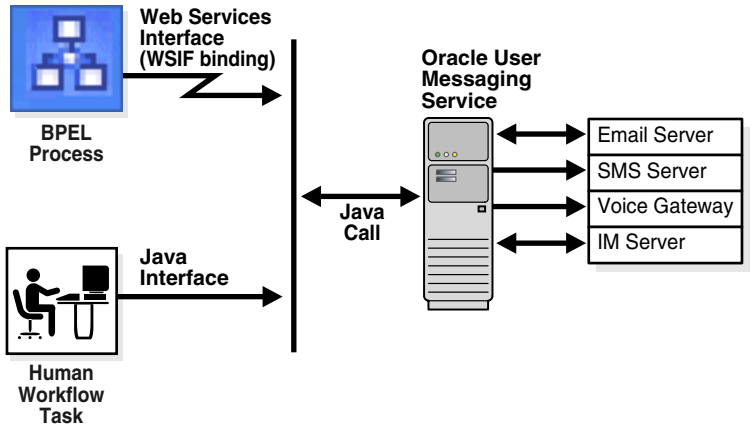
  Oracle User Messaging Service is a new feature for release 11*g*. The BPEL notification service uses the underlying infrastructure provided by Oracle User Messaging Service to send notifications.

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

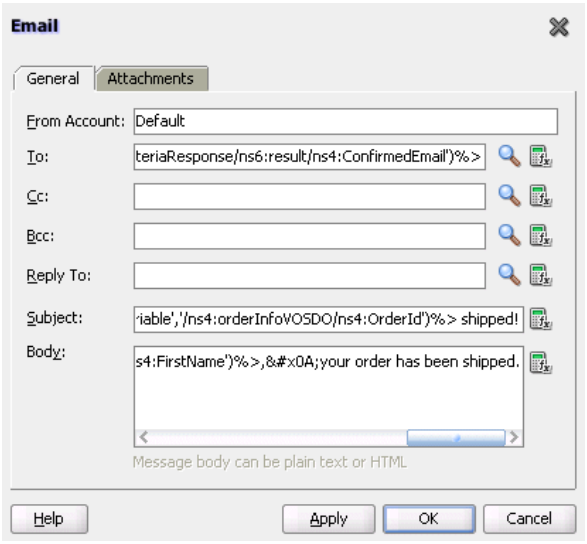
If You Selected...	See...
Email	Section 16.3.1, "How To Configure the Email Notification Channel" to configure email notification
IM	Section 16.3.2, "How to Configure the IM Notification Channel" to configure IM notification
SMS	Section 16.3.3, "How to Configure the SMS Notification Channel" to configure 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 *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.
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 *n* 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:findCustome
     rInfoVO1CustomerInfoVOCriteriaResponse/ns3:result/ns2:ConfirmedEmail')
     ```

   - A combination of manually-entered text and an XPath expression specifies the ID of the order:

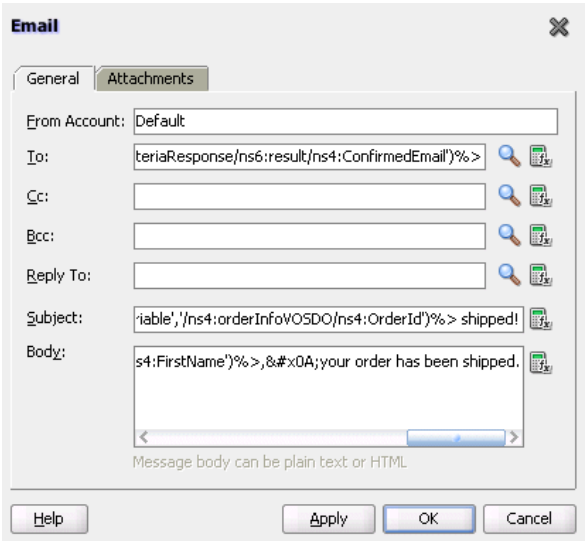
     ```
     Order with id
     <%bpws:getVariableData('gOrderInfoVariable','/ns2:orderInfoVOSDO/ns2:OrderI
     d')%> shipped!
     ```

   - A combination of manually-entered text and an XPath expression specifies the body of the email message:

     ```
     Dear
     <%bpws:getVariableData('gCustomerInfoVariable','parameters','/ns6:findCusto
     merInfoVO1CustomerInfoVOCriteriaResponse/ns6:result/ns4:FirstName')%>,
     your order has been shipped.
     ```

   Figure 16–3 provides details.

Figure 16–3 Email Dialog



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:

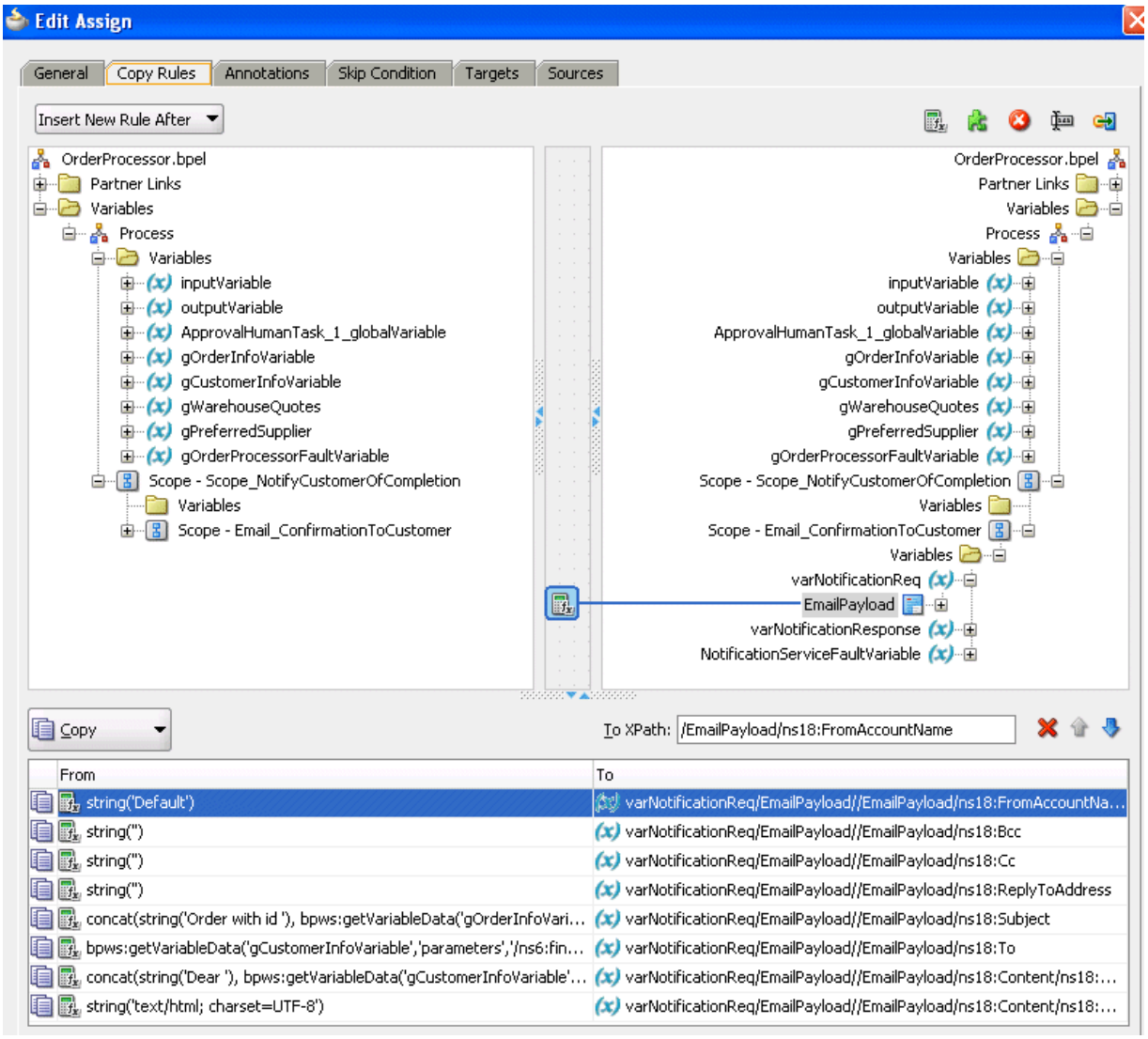
1. From the Component Palette, select **Email** as the notification channel.
2. Specify values for **To**, **Subject**, and **Body**.
3. Click the **Attachments** tab.
4. Click the **Add** icon to add as many attachments as you require. (Note that the number of attachments does not need to include the 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.

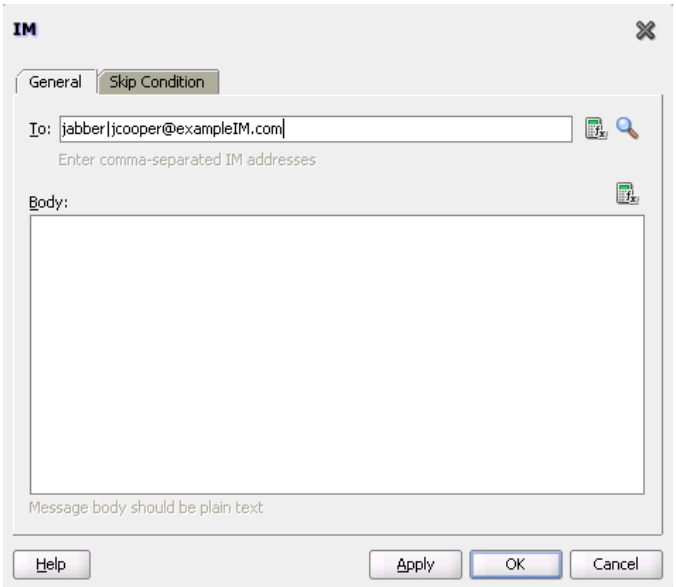
Example 16–3 CDATA() Function

```
<%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**.

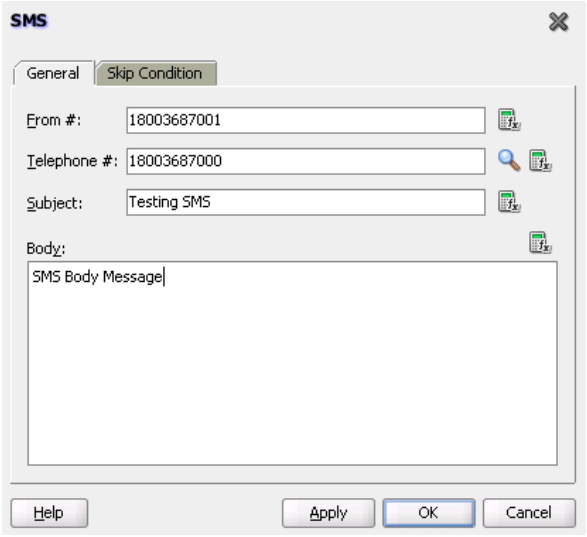
   The BPEL fragment that invokes the notification service for IM notification is created.

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**.

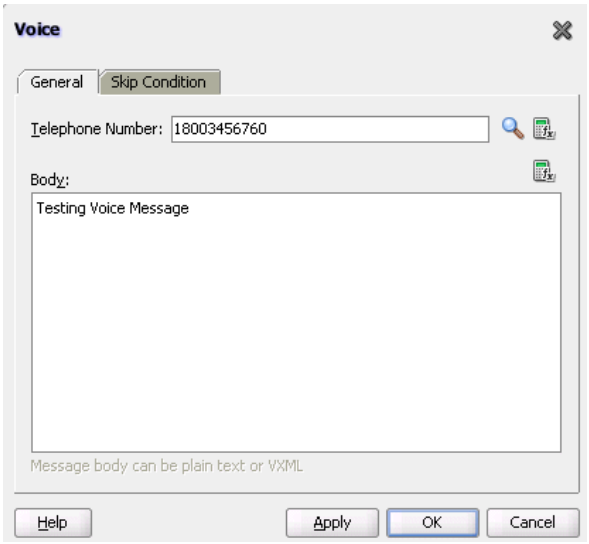
   The BPEL fragment that invokes the notification service for SMS notification is created.

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."

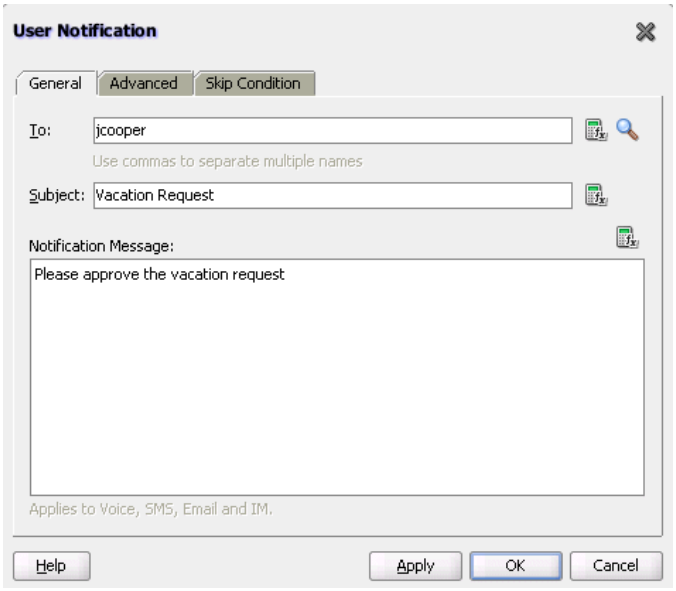
For more information about the User Messaging Preferences user interface, see Chapter 64, "User Messaging Preferences."

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



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"
```

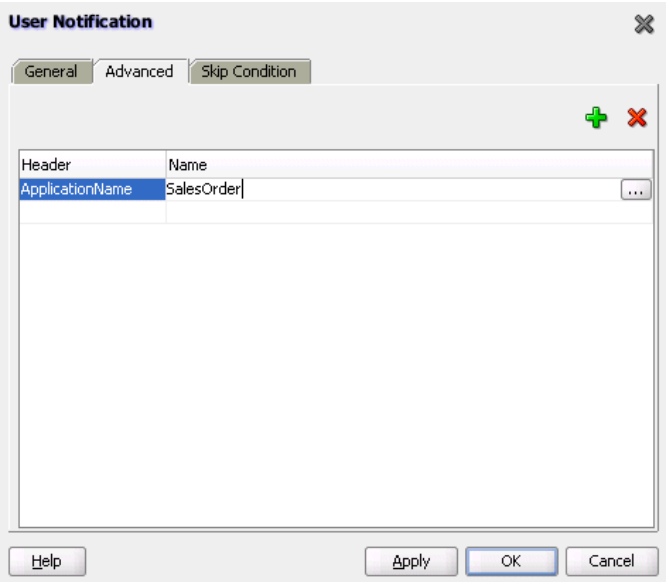
- The end user `jstein` creates the following preference rule in the User Messaging Preferences user interface:

```
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**.

17

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

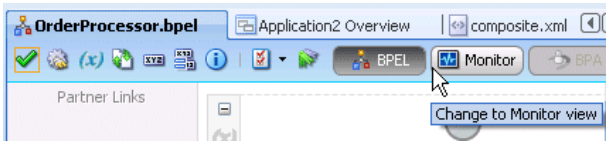
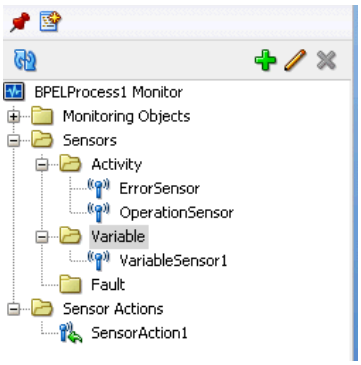


Figure 17–2 shows the sensor actions and sensors in the Structure window.

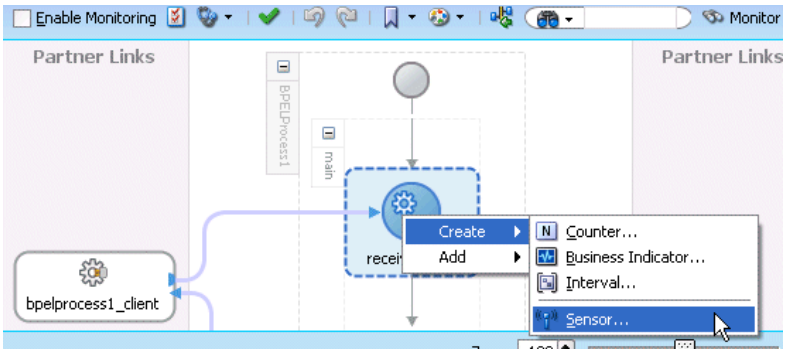
Figure 17–2 Sensors and Sensor Actions Displayed in Oracle JDeveloper



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:

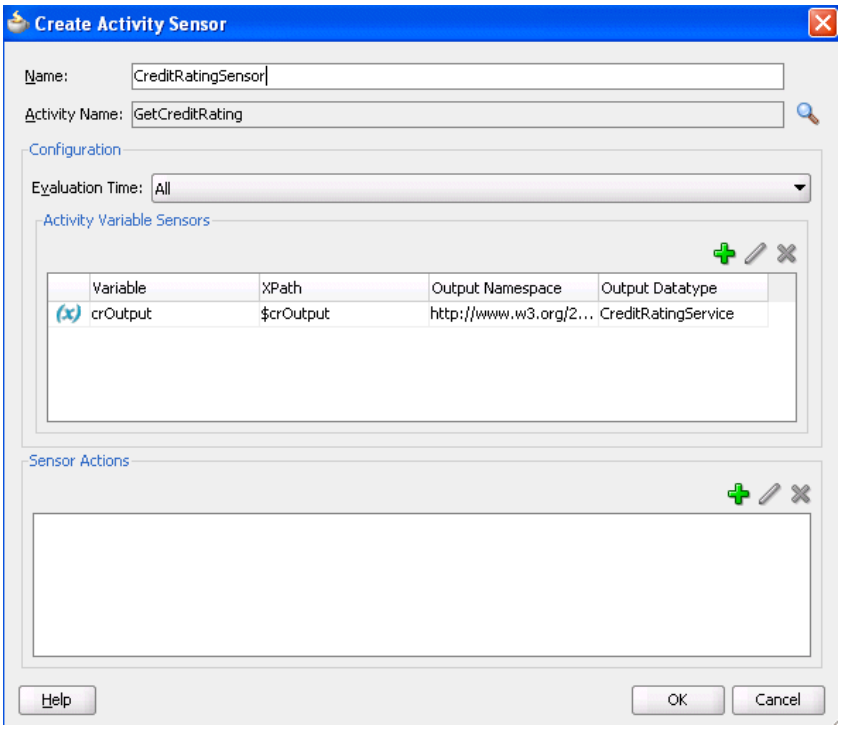
Assume you are monitoring a loan flow application, and want to know the following:

- 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>
```

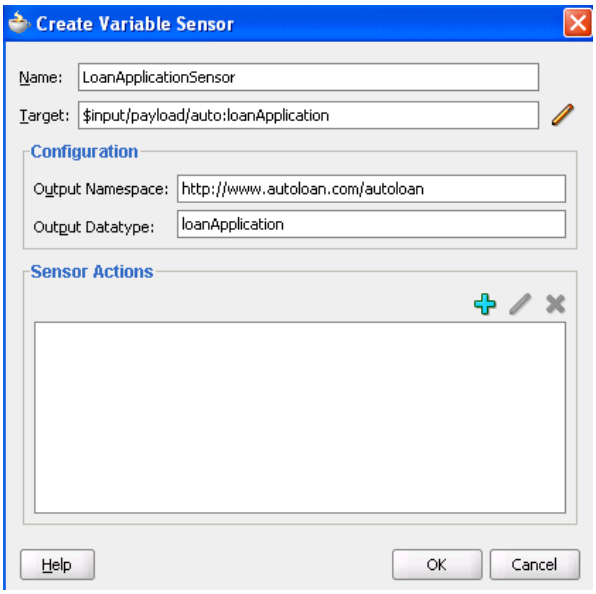
6. If you want to create a variable sensor on the activity, then in the **Activity Variable Sensors** section, click the **Add** icon. This is an optional field.

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

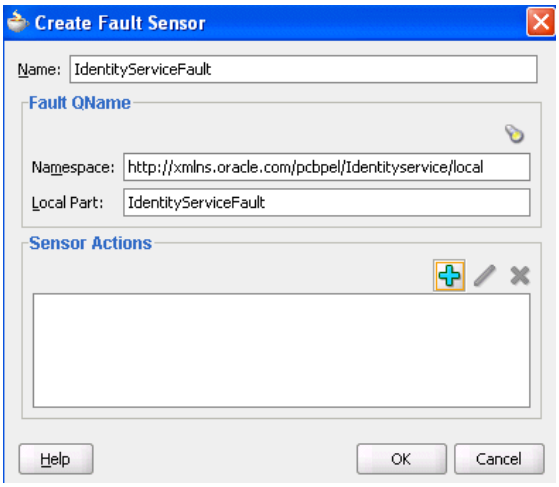
```
<sensor sensorName="LoanApplicationSensor"
    classname="oracle.tip.pc.services.reports.dca.agents.BpelVariableSensorAgent"
    kind="variable"
    target="$input/payload">
  <variableConfig outputNamespace="http://www.autoloan.com/ns/autoloan"
                  outputDataType="loanApplication"/>
</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.

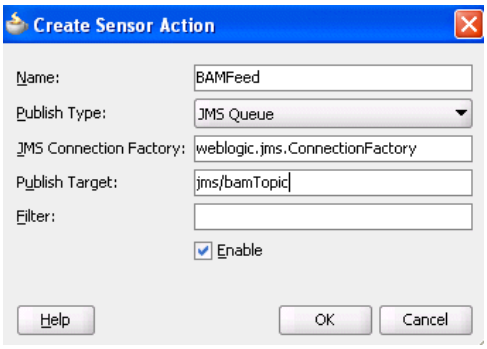
Table 17–1 Sensor Actions Dialog

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 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:

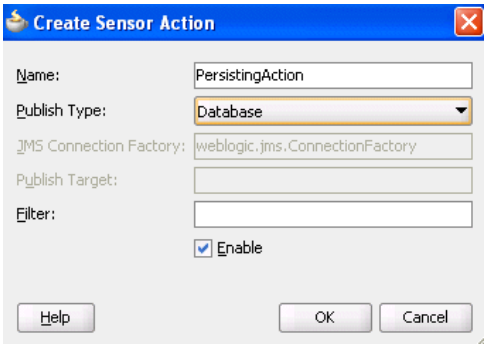
Example 17–4 bpel_process_name_sensorAction.xml file

```
<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:

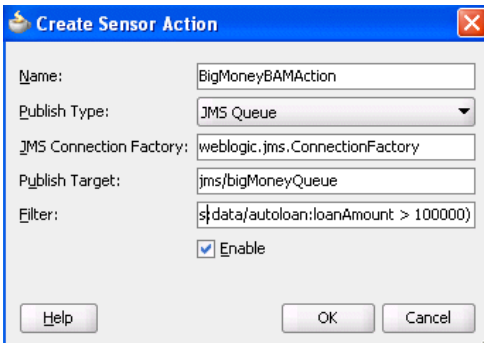
Example 17–5 bpel_process_name_sensorAction.xml file

```
<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

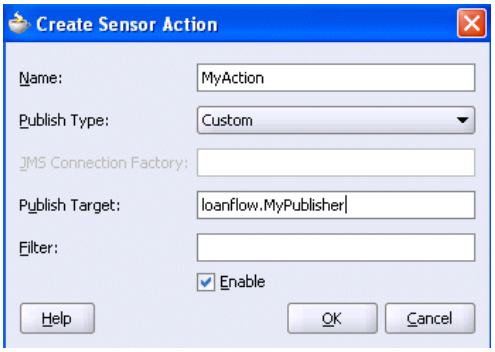
```
<action name="BigMoneyBAMAction"
        enabled='true'
        filter="boolean(/s:actionData/s:payload
                        /s:variableData/s:data
                        /autoloan:loanAmount > 100000)"
        publishType="JMSQueue"
        publishTarget="jms/bigMoneyQueue">
  <sensorName>LoanApplicationSensor</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."

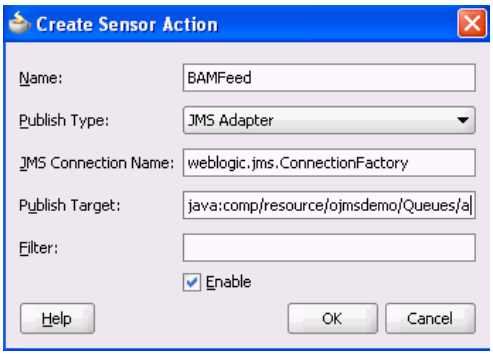
Figure 17–10 Using the Custom Publish Type



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.

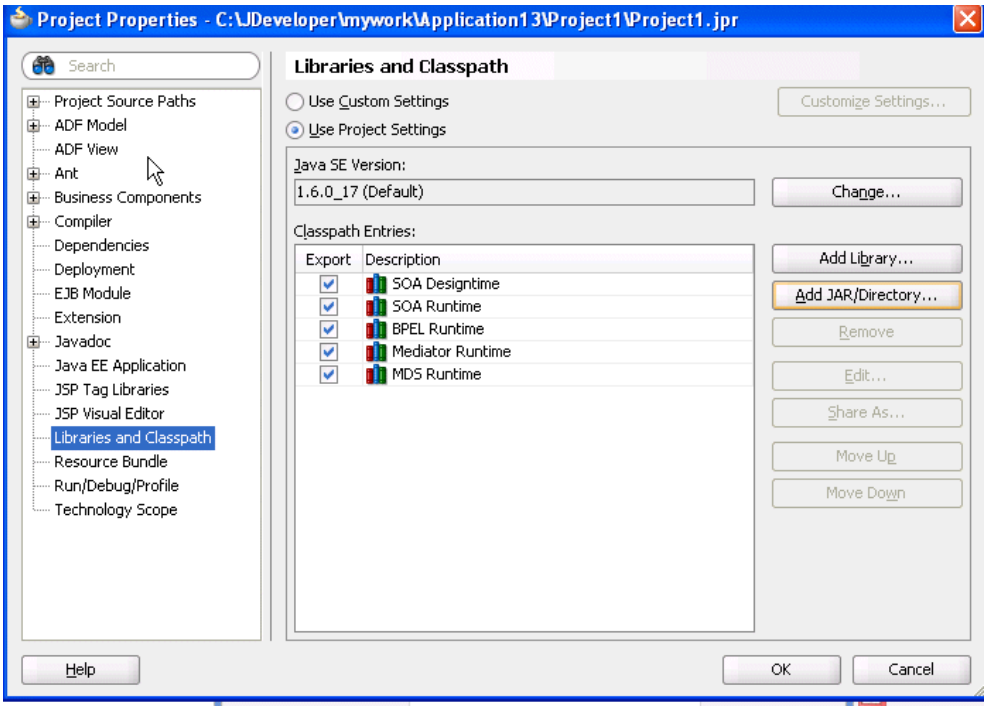
2. Click **Libraries and Classpath**.

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:

18

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:

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 an process messages consisting of multiple parts. Some Remote Procedure Call (RPC) web services contain multiple parts in the SOAP message.

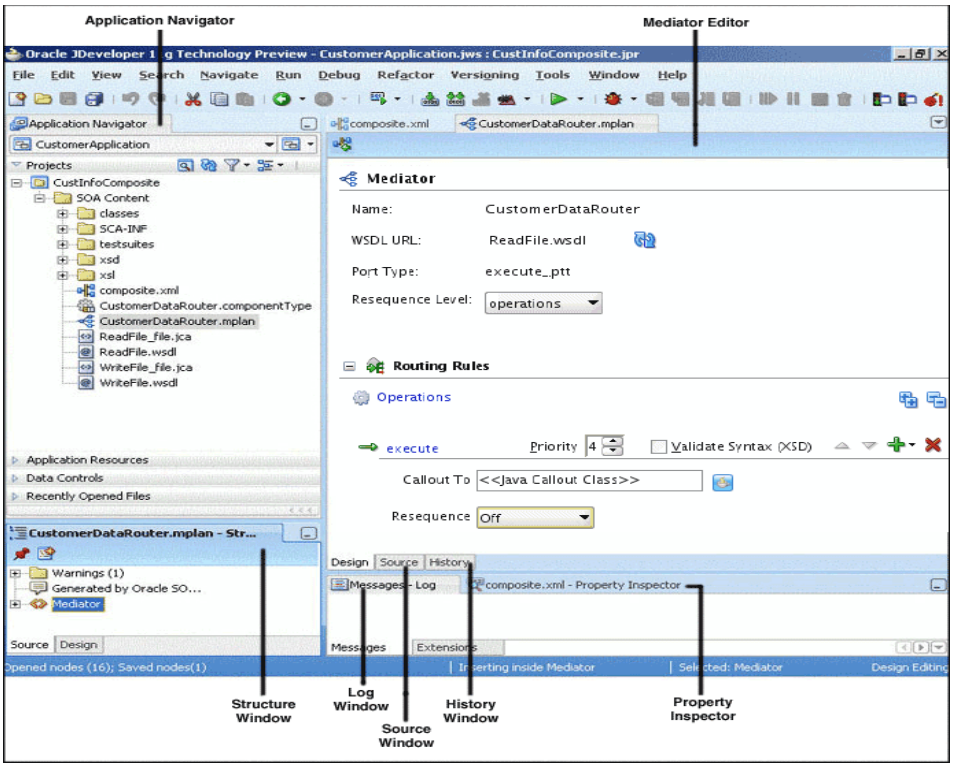
  For more information about multiple part message support, see Chapter 20, "Working with Multiple Part Messages in Oracle Mediator."

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:

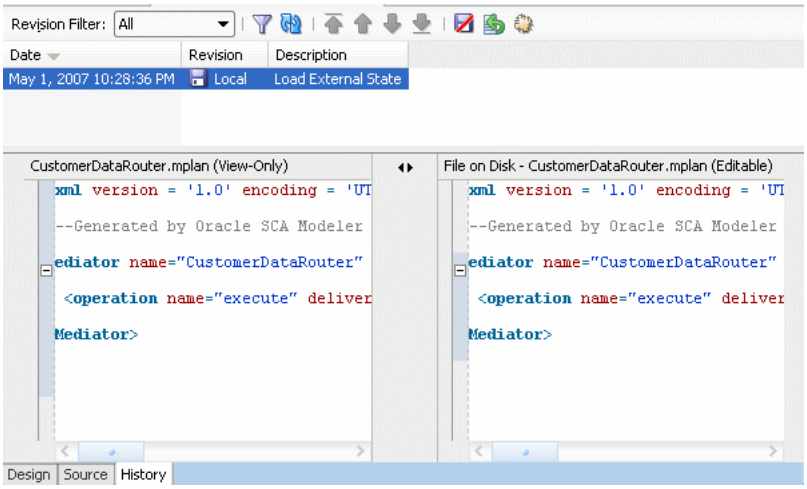
Example 18–1 Oracle Mediator Source Code

```
<?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:

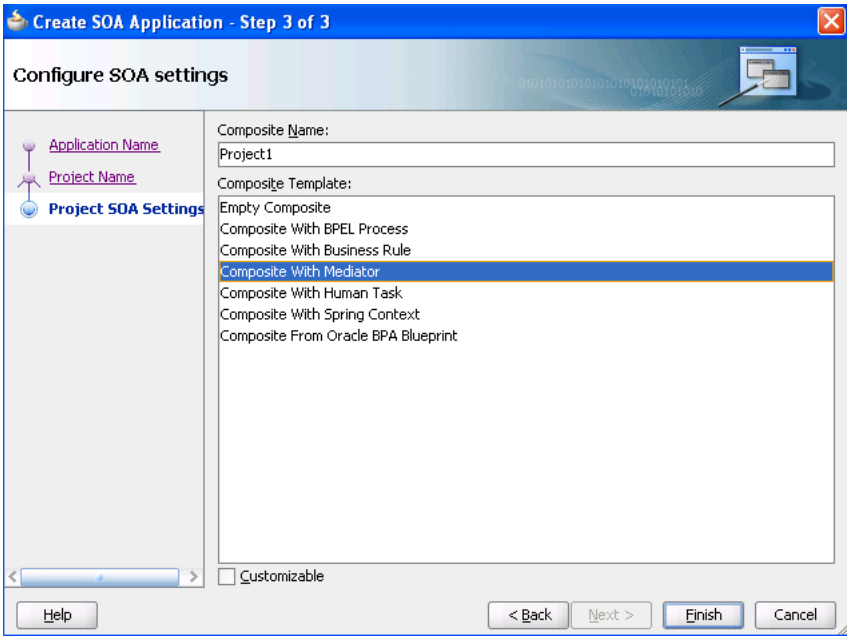
- When you create a composite application
- From within an existing composite application
- When you create a project
- From within an existing project

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.

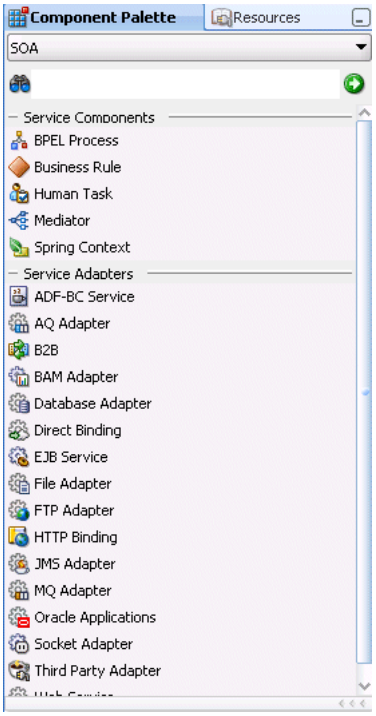
4. Configure the Oracle Mediator interface, as described in Section 18.4, "Configuring the Oracle Mediator Interface Definition".

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".

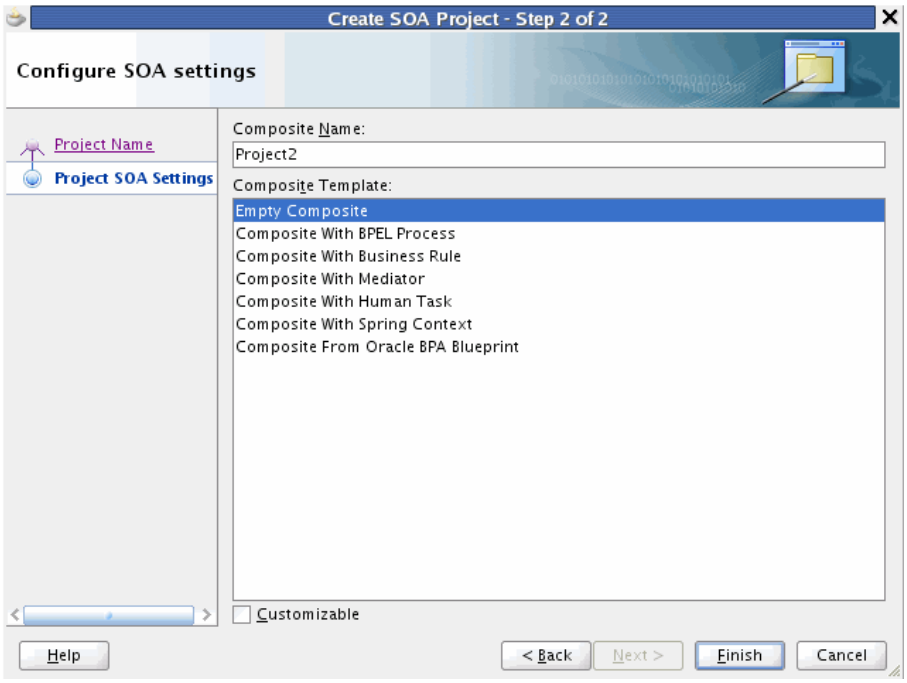
To create an Oracle Mediator when creating a new project:

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**.

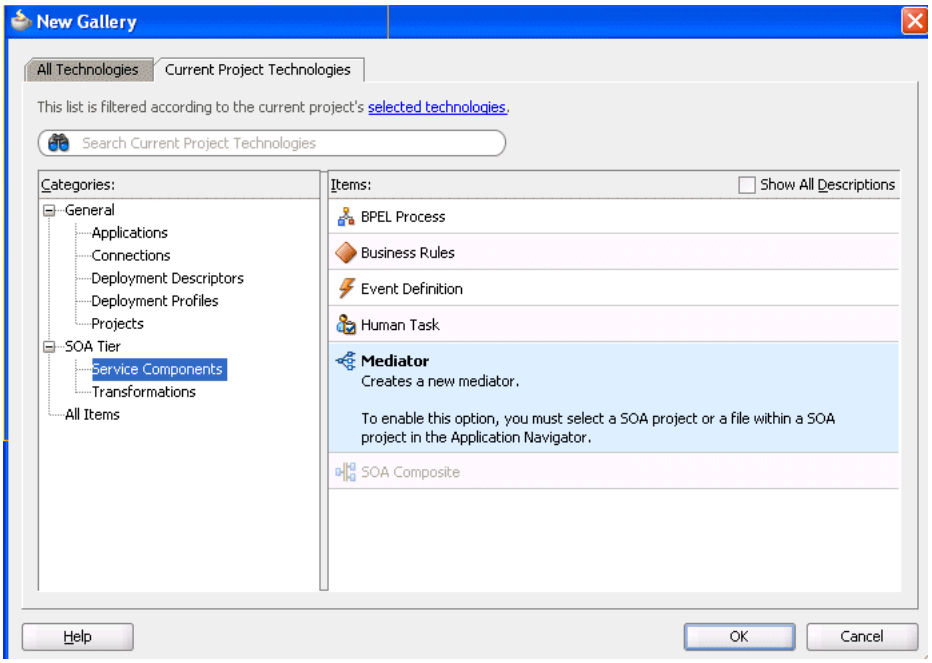
   The Create Mediator dialog appears.

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.

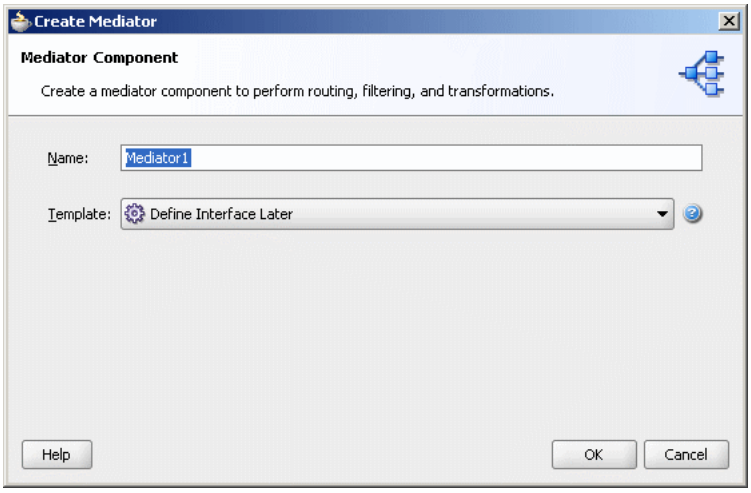
To create an Oracle Mediator without an interface definition:

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

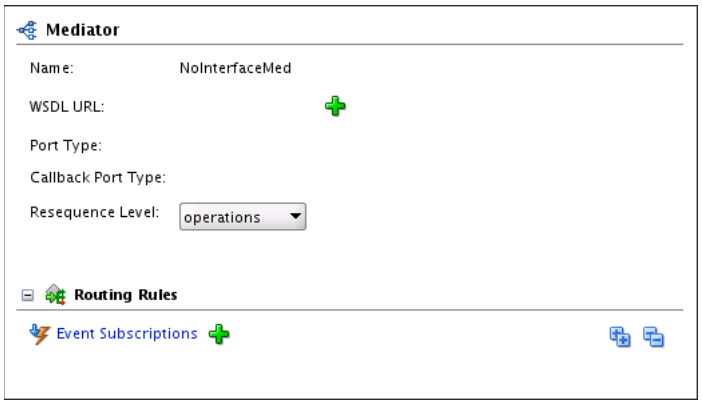


18.4.1.2 What Happens When You Create an Oracle Mediator Without an Interface Definition

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:

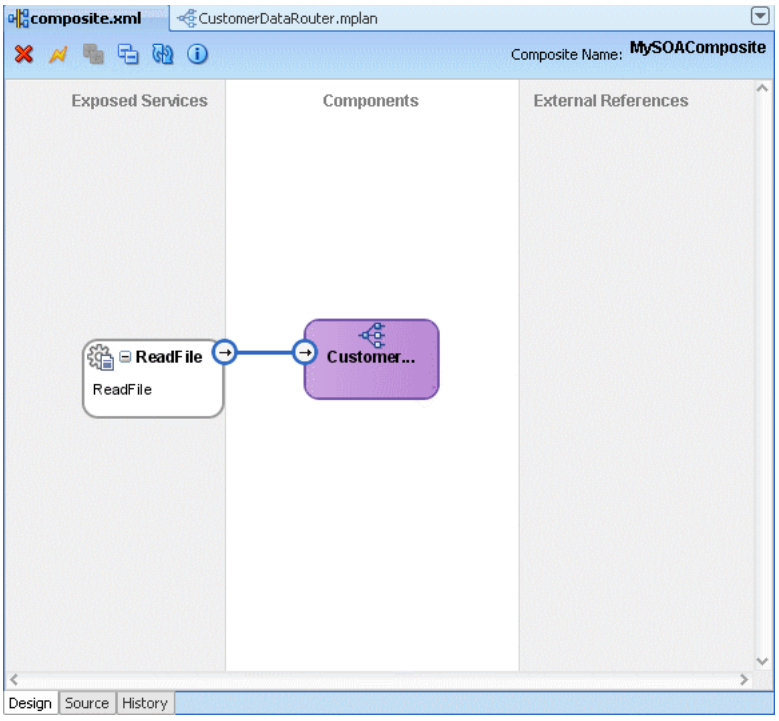
- In the SOA Composite Editor, drag a wire from an Oracle Mediator to a service.

  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

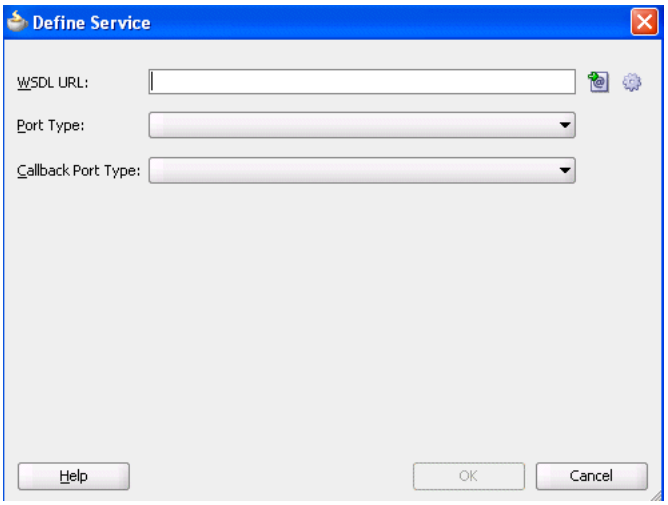


To define services for an Oracle Mediator in the Mediator Editor:

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

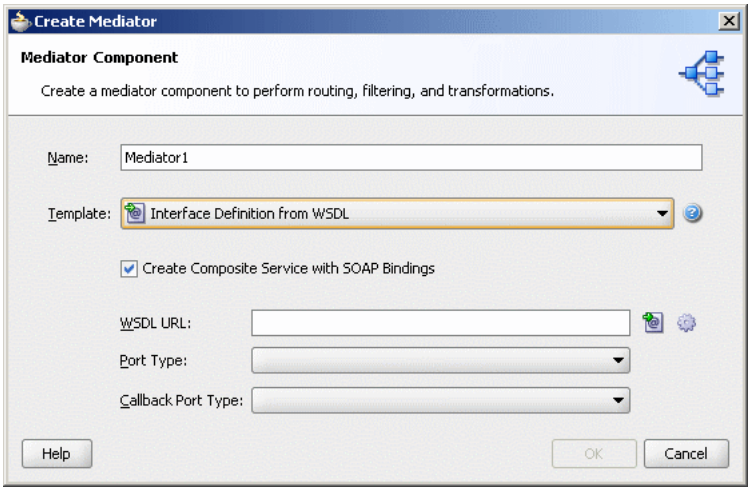
The **Interface Definition from WSDL** template on the Create Mediator dialog box creates an Oracle Mediator based on a WSDL file that has already been created or that can be generated from a schema.

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.

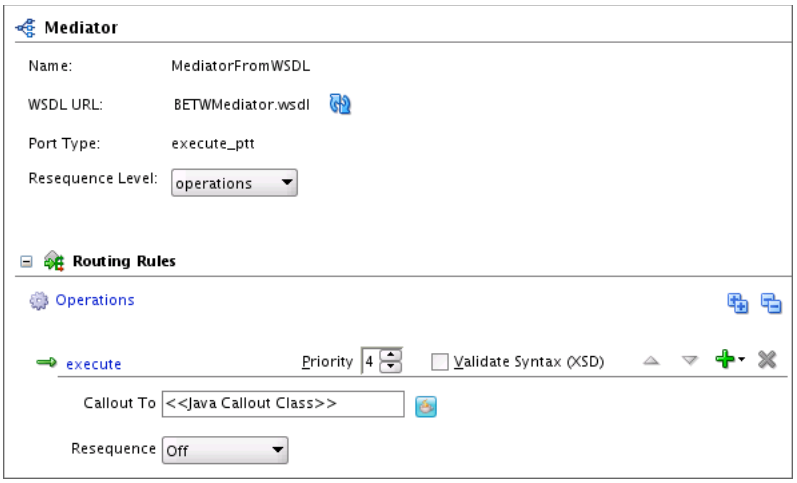
   A callback port is the one to which the response message is sent in an asynchronous communication.
8. Click **OK**.

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



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.

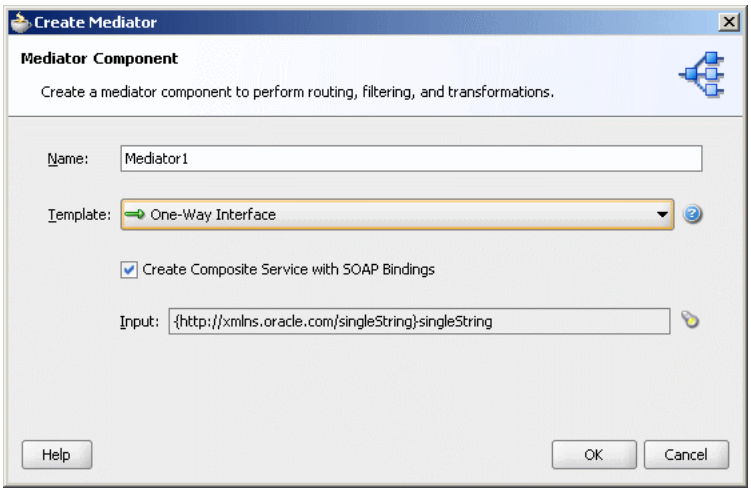
To create an Oracle Mediator with a one-way interface definition:

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



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>
```

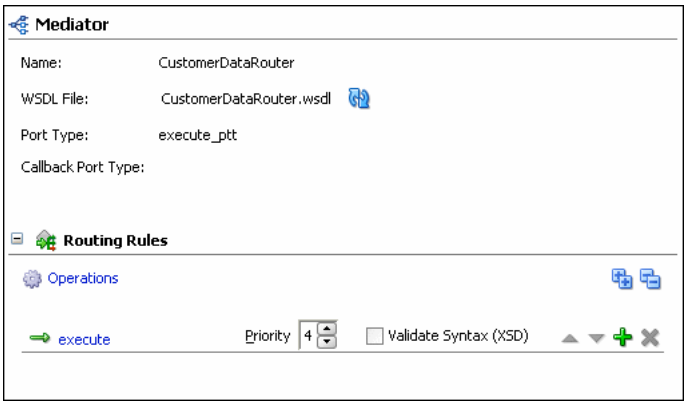
6. Click **OK**.

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.

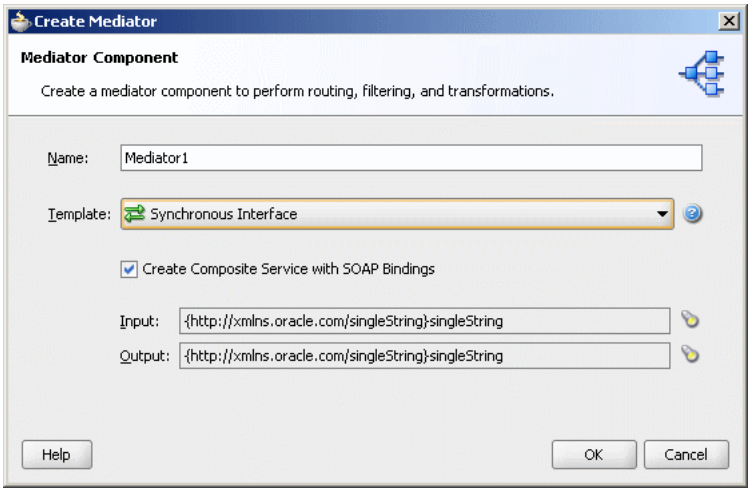
To create an Oracle Mediator with a synchronous interface definition:

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**.

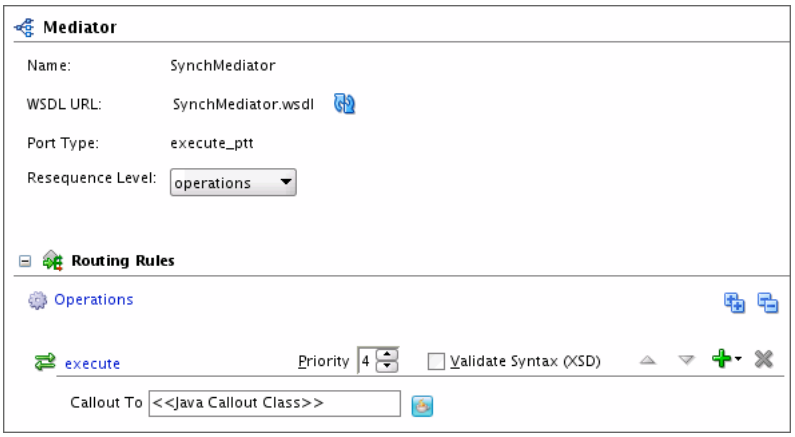
18.4.4.2 What Happens When You Create an Oracle Mediator with a Synchronous Interface Definition

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.

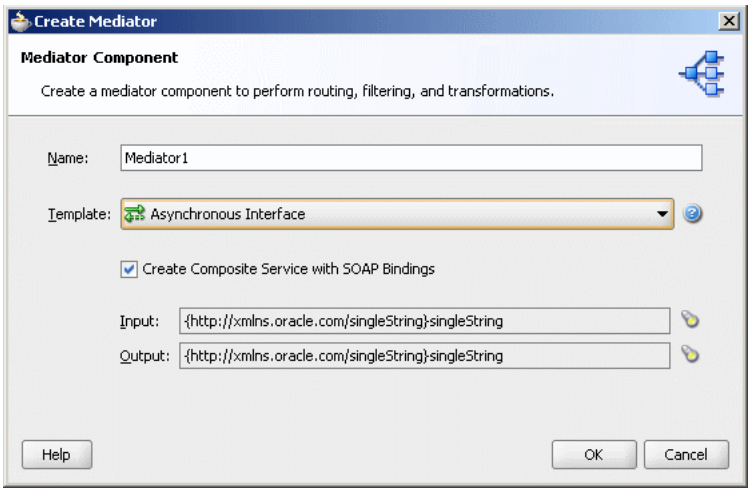
To create an Oracle Mediator with an asynchronous interface definition:

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.

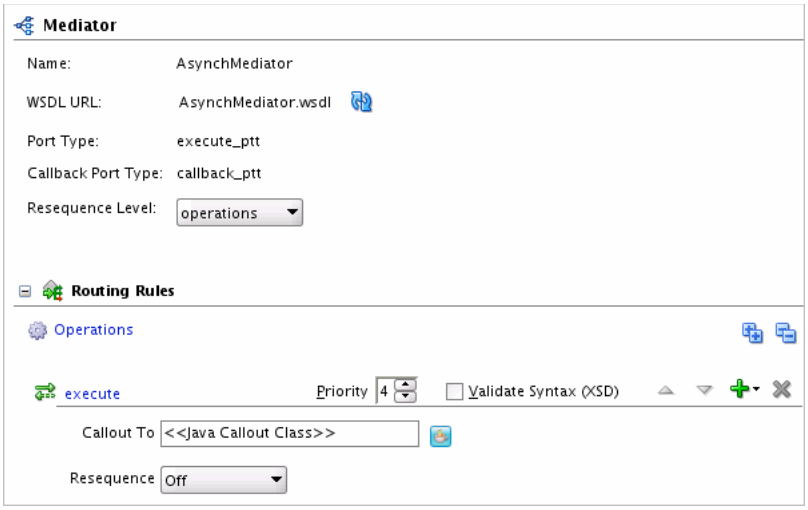
   By default, the **singleString** schema element is selected for the output message.
7. Click **OK**.

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



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.

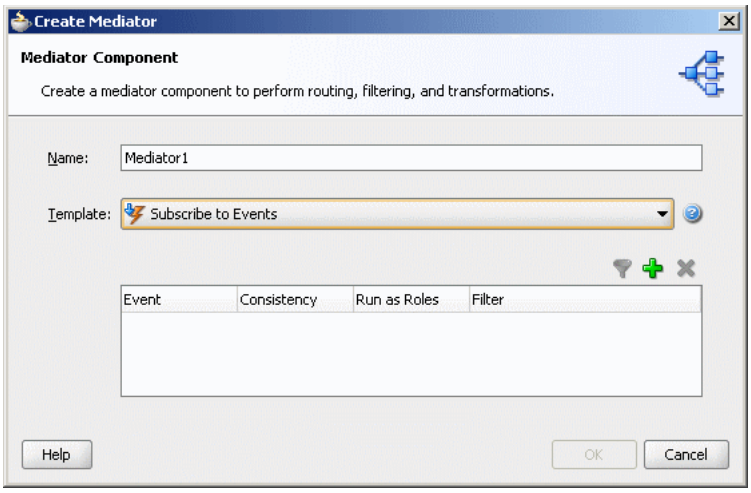
To create an Oracle Mediator for an event subscription:

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**.

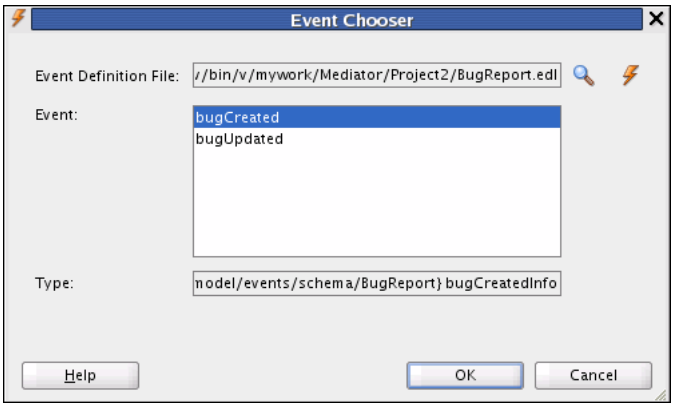
   The SOA Resource Browser dialog appears.

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.

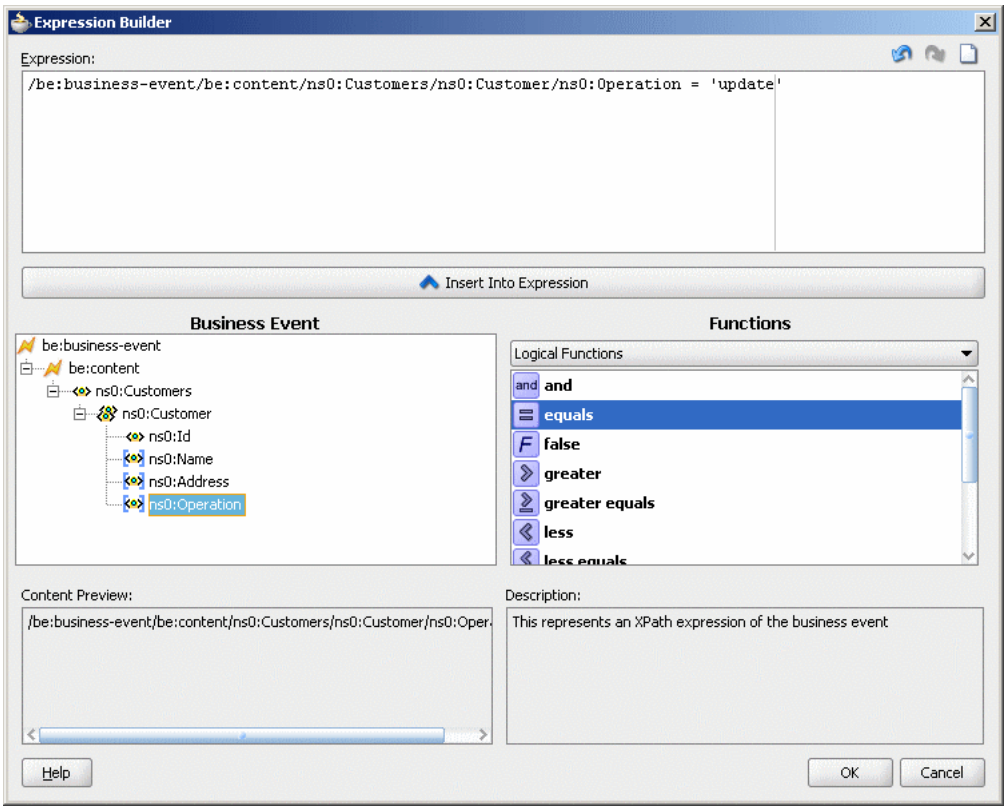
10. To filter the event, perform any of the following:
    - Double-click the **Filter** column of the selected event.
    - Select the event and then click the **filter** icon (first icon).

    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.

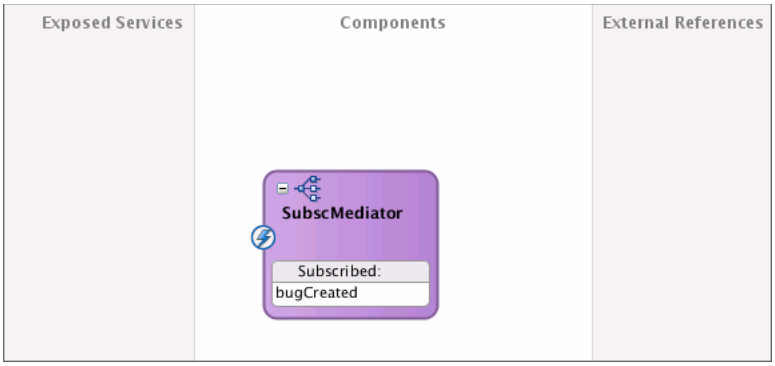
12. Click **OK**.

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.

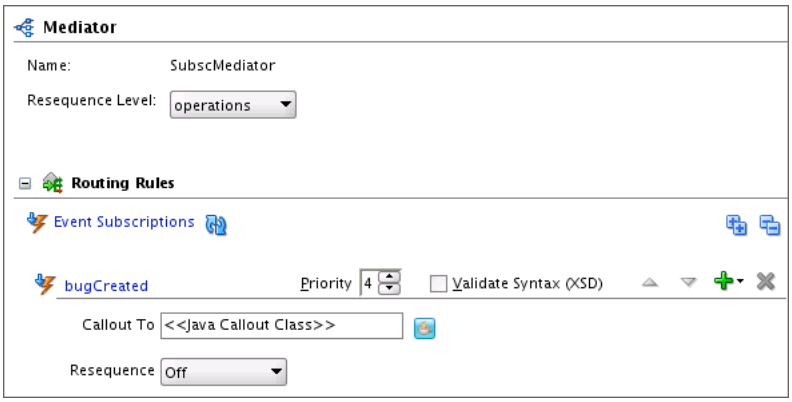
When you view the Oracle Mediator component in the SOA Composite Editor, the icon on the left side of the Oracle Mediator indicates that this Oracle Mediator is configured for an event subscription, as shown in Figure 18–22.

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.

1. 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`.

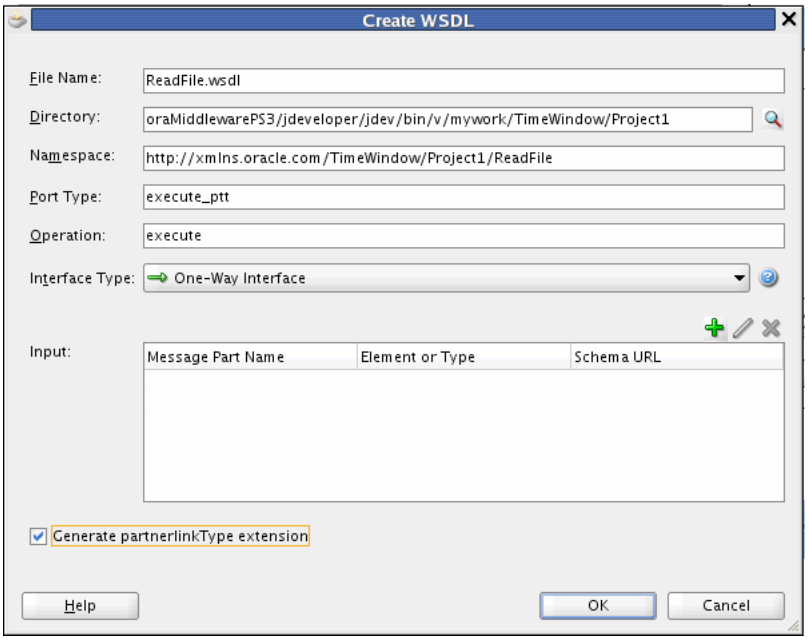
     The namespace that you specify is defined as the `tns` namespace in the WSDL file.
   - **Port Type**: The name of the port type in the WSDL file that contains the operation to use.
   - **Operation**: The name of the action to perform; for example, `executeQuery`.

   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.

2. In the **Interface Type** field, select **One-Way Interface**.

   The **Input** field appears, as shown in Figure 18–24.

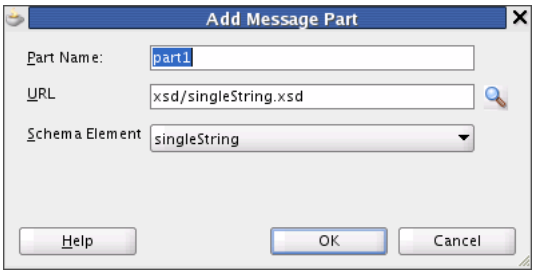
Figure 18–24 Create WSDL Dialog for a One-Way Interface



3. 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.

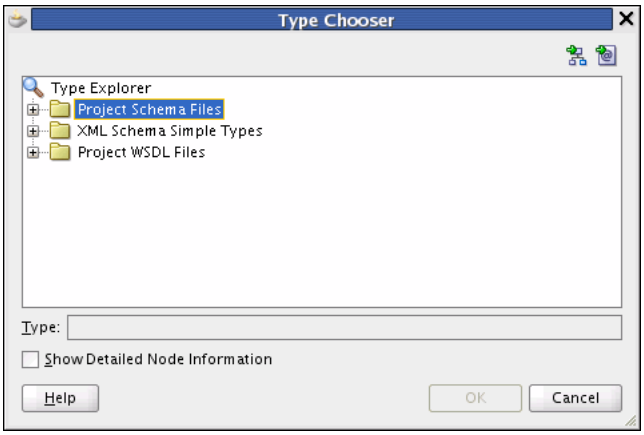
Figure 18–25 Add Message Part Dialog



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`.

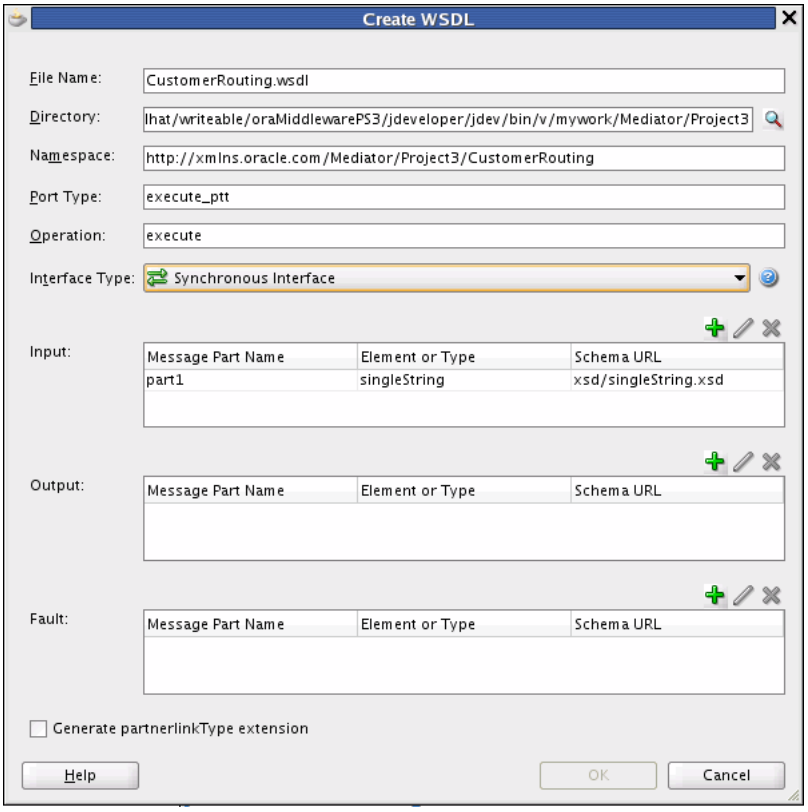
     Note: Spaces and special characters are not allowed in an operation name or port type. Only alphabetic and numeric characters are supported, and the first character cannot be a number.

     ---

2. In the **Interface Type** field, select **Synchronous Interface**.

   The **Input**, **Output**, and **Fault** fields appear, as shown in Figure 18–27.

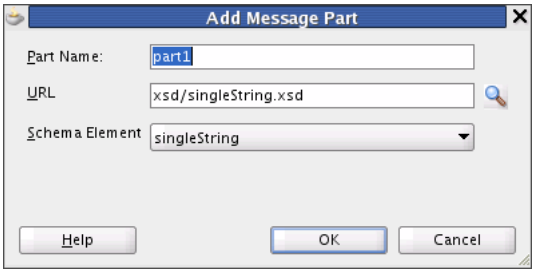
Figure 18–27 Create WSDL Dialog for a Synchronous Interface



3. 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.

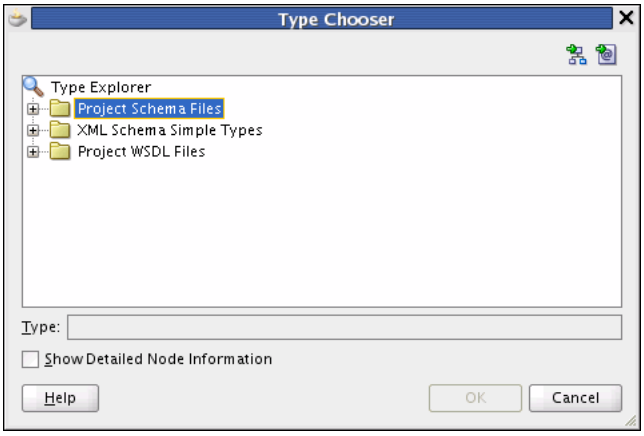
Figure 18–28 Add Message Part Dialog



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 18–29.

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.

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`.

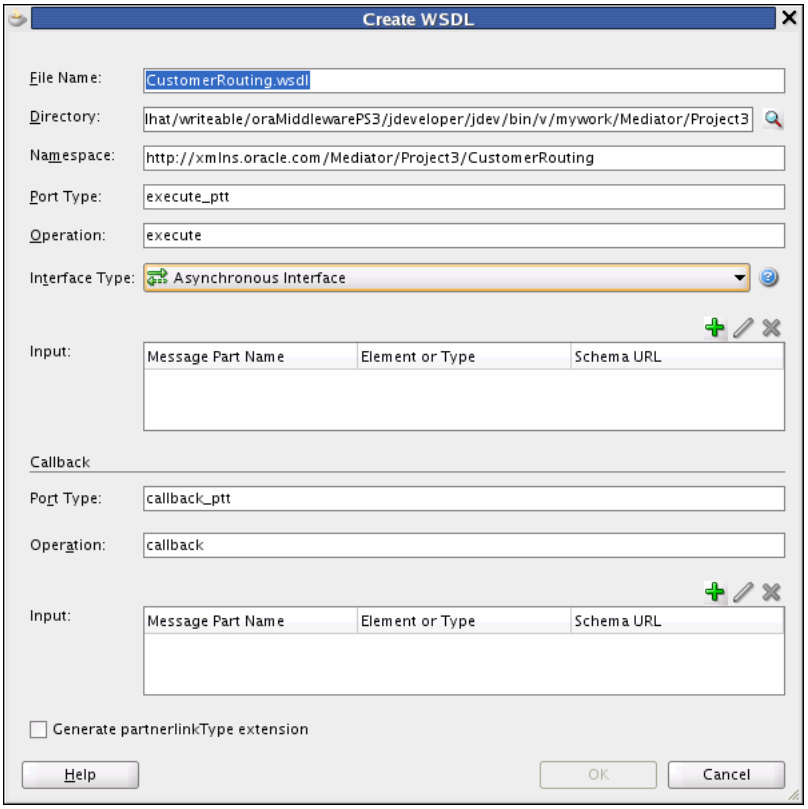
     The namespace that you specify is defined as the `tns` namespace in the WSDL file.
   - **Port Type**: The name of the port type in the WSDL file that contains the operation to use.
   - **Operation**: The name of the action to perform; for example, `executeQuery`.

   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 **Asynchronous Interface**.

   The **Input** field and **Callback** section appear, as shown in Figure 18–30.

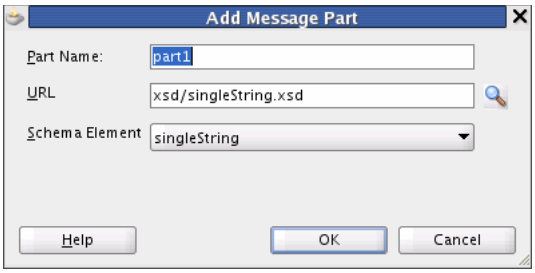
Figure 18–30 Create WSDL Dialog for an Asynchronous Interface



3. 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.

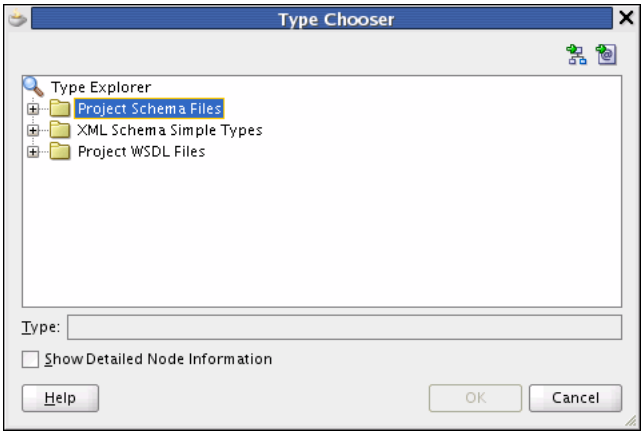
Figure 18–31 Add Message Part Dialog



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

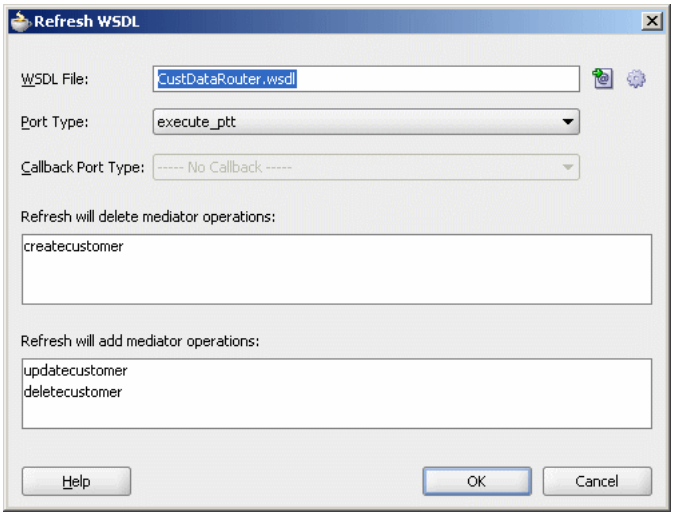
You can modify an Oracle Mediator WSDL file by adding or deleting operations. After modifying the WSDL file, use the Refresh WSDL dialog to synchronize the changes.

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

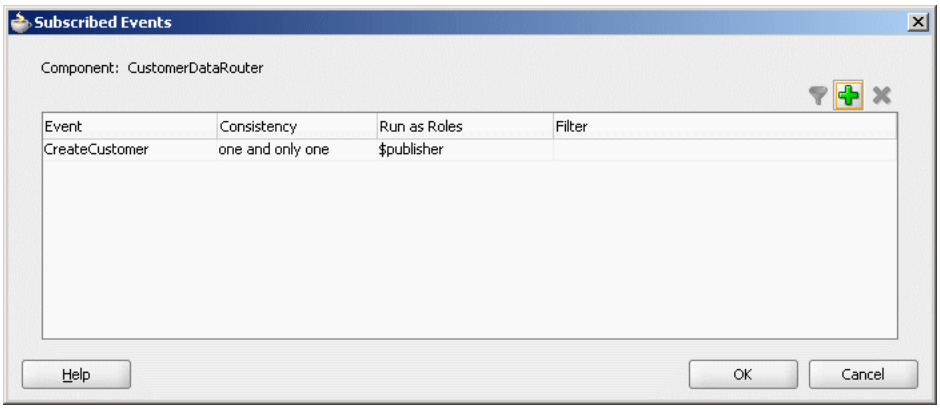
You can subscribe to new events, modify existing event subscriptions, and unsubscribe from subscribed events using the **Manage Event Subscriptions** option in the Mediator Editor.

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**.

19

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

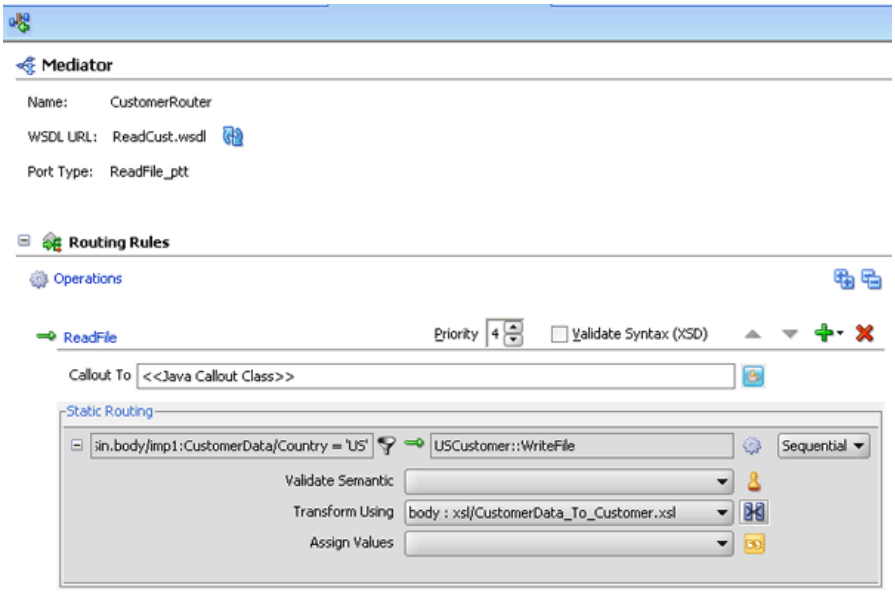


Figure 19–2 lists and describes the icons in the **Routing Rules** section.

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.

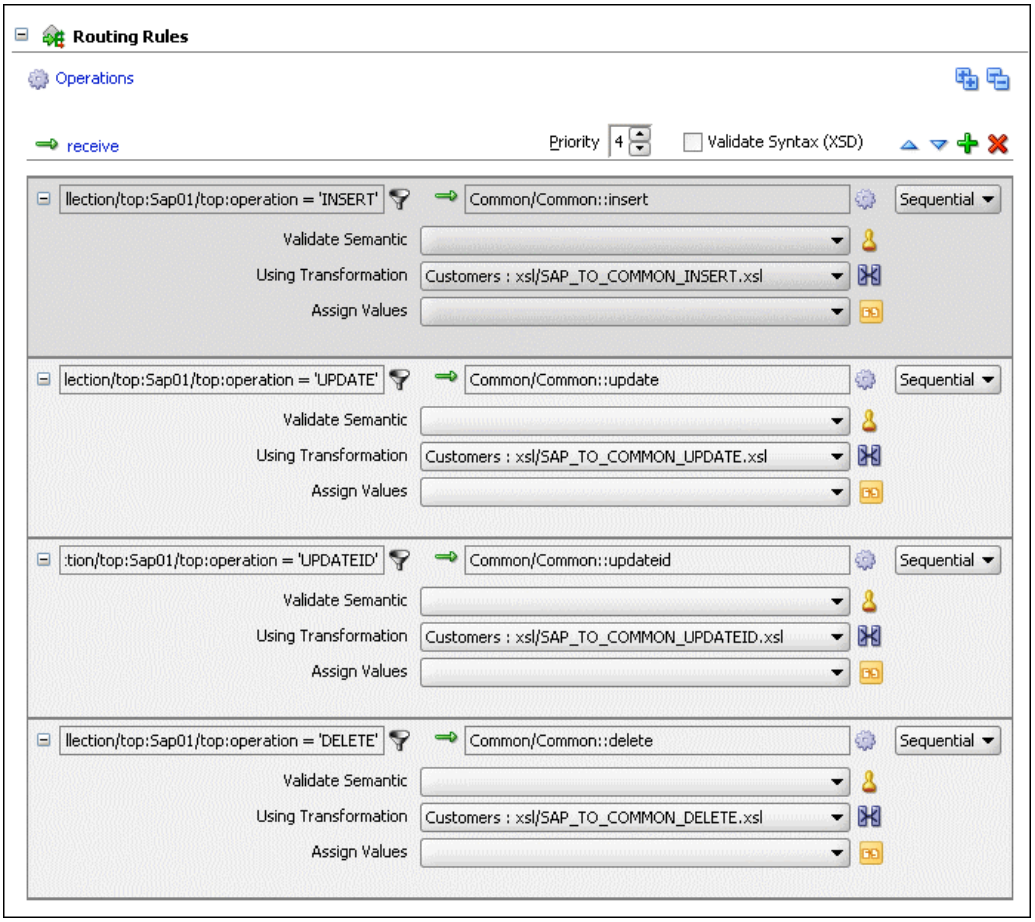
You can specify multiple routings for an inbound operation or event. Each routing is mapped to one target service invocation or event. Therefore, to specify multiple

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**.

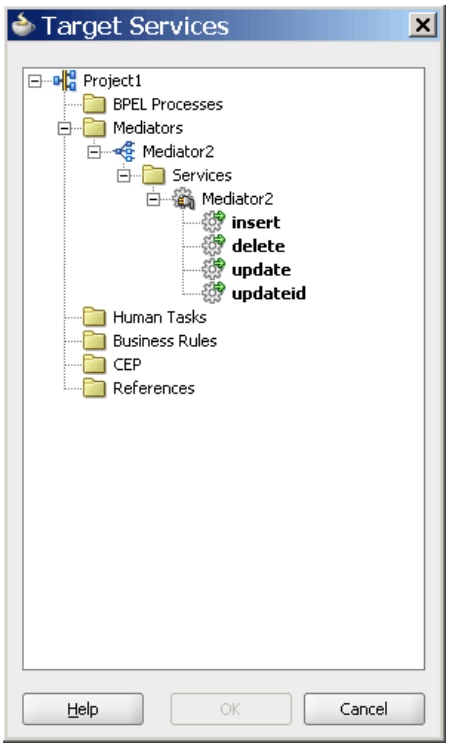
   The Target Type dialog appears, as shown in Figure 19–4.

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.

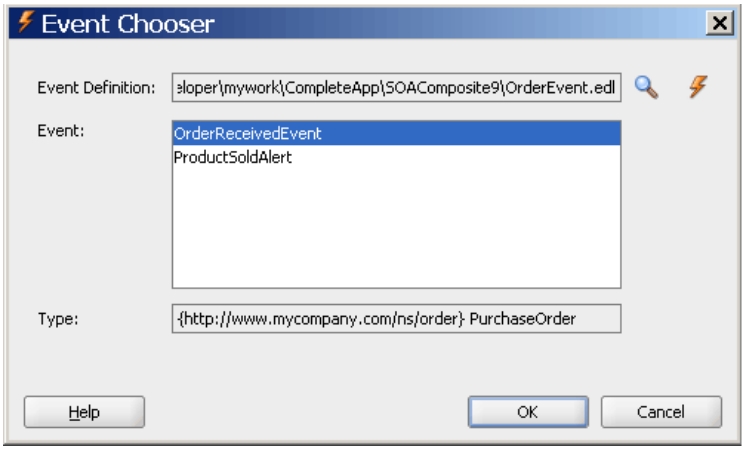
3. To the right of the **Event Definition** field, click **Search**.

   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.

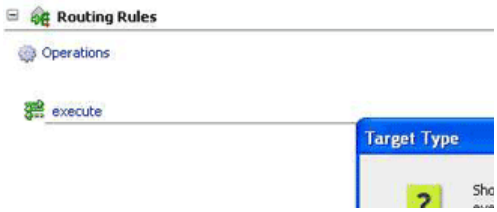
7. Configure the routing rule as described the remaining sections of this chapter.

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

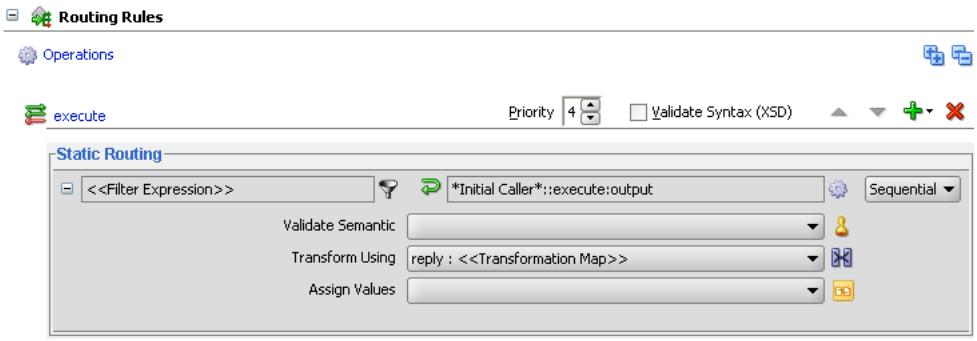


2. Click **Echo**.

> **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.

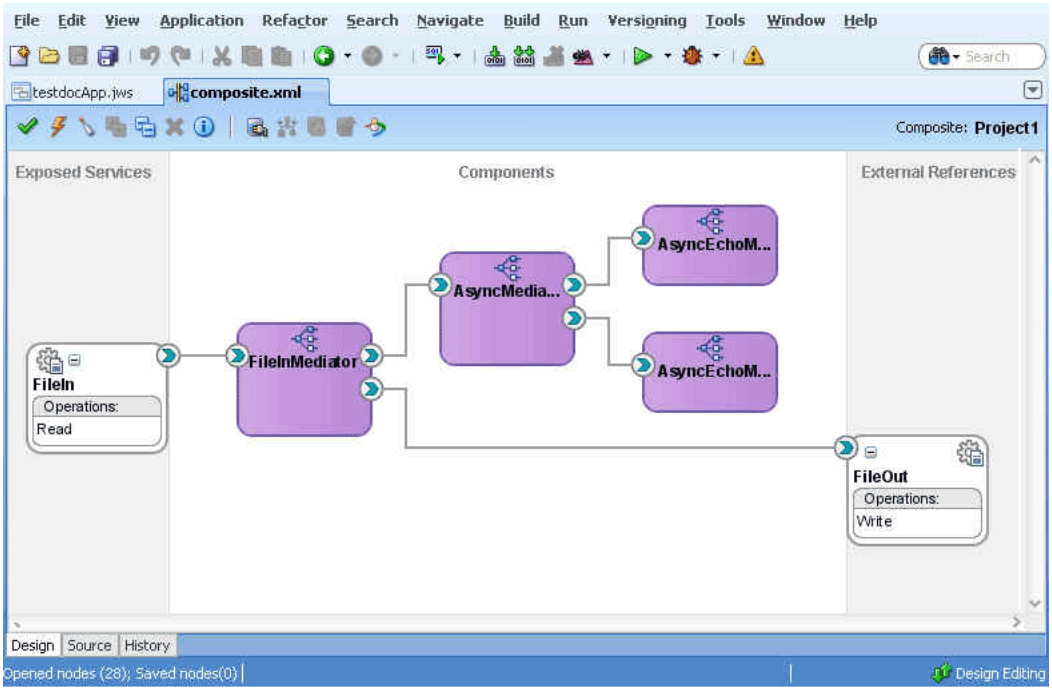
Note: If the number of routing rules is larger and the time taken to execute the routing rules exceeds the transaction timeout, you must set the transaction timeout to a value that is greater than the time taken to execute all the routing rules.

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

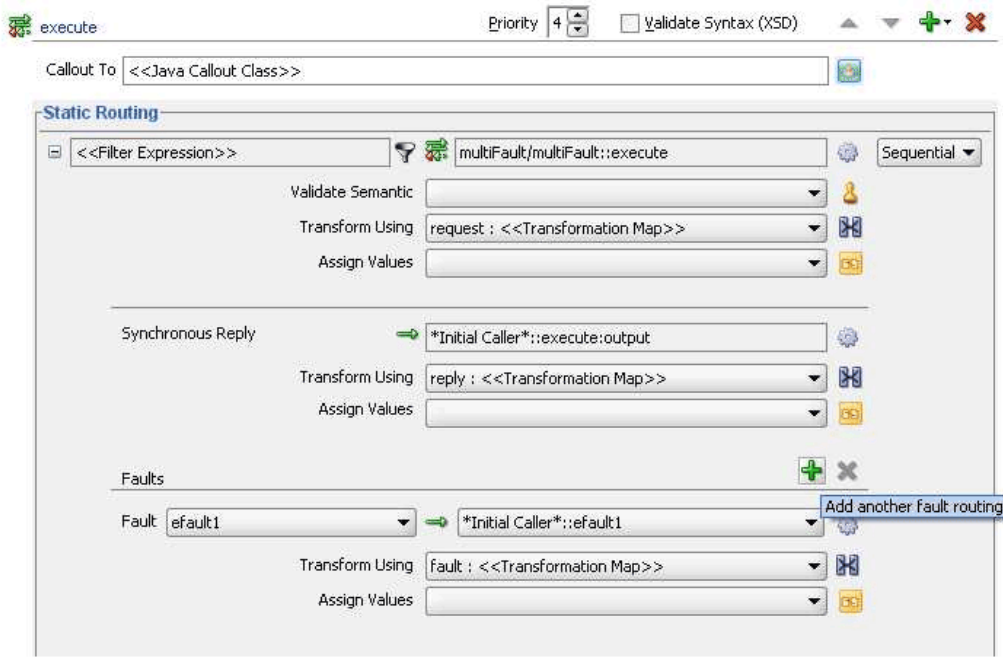
If you create a new routing rule in which the target service operation has one or more faults, you still see a single fault routing section in the Mediator Editor. If the source Oracle Mediator service component supports one or more faults, then the fault is routed back to the caller by default. You 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.

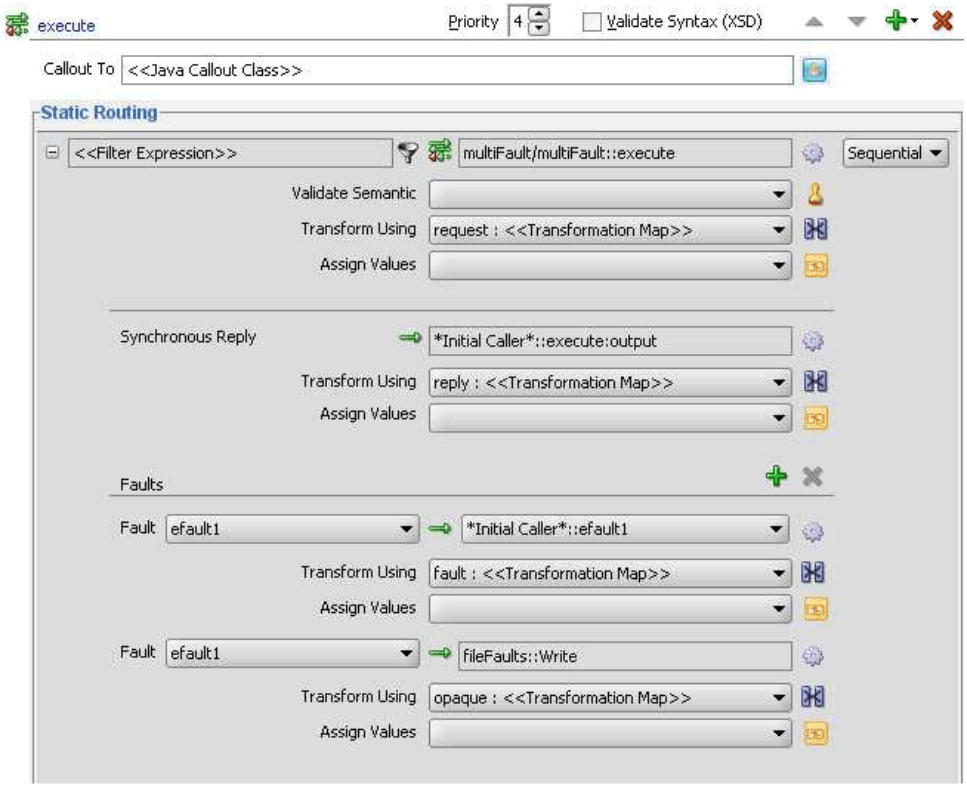
Figure 19–10 Adding a Second Fault



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



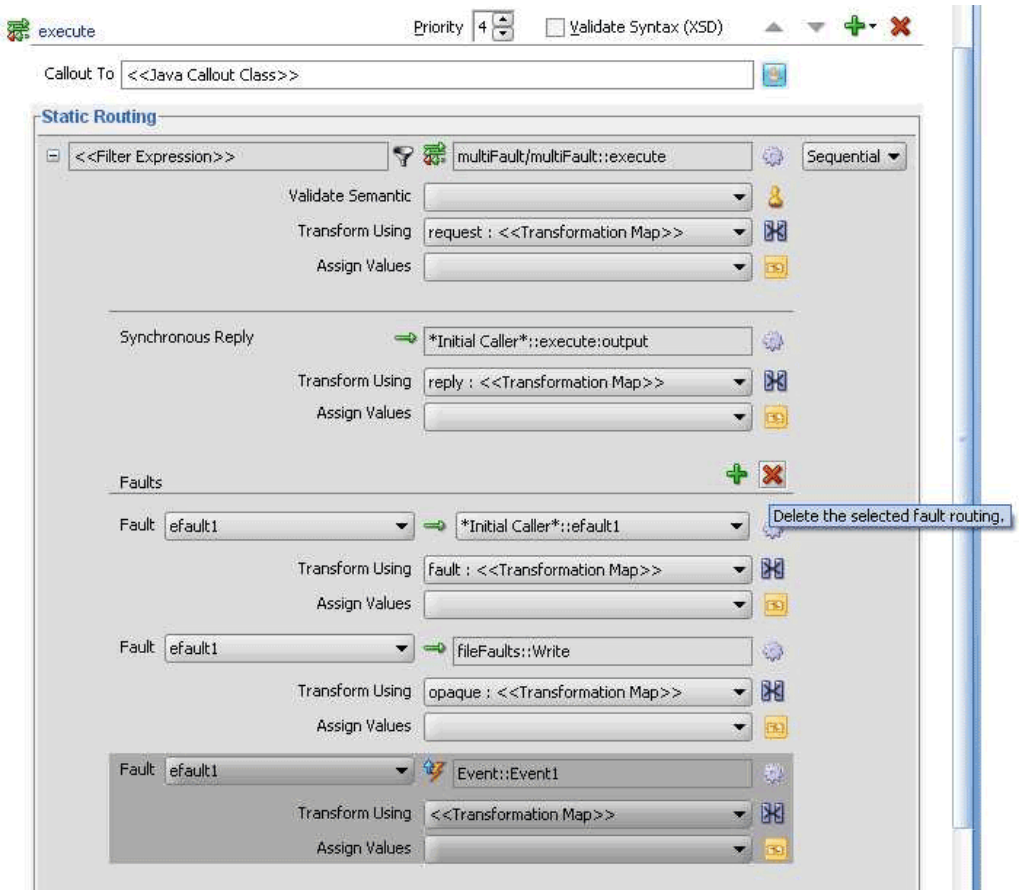
Note: You can route the same fault to multiple targets using different transformations.

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"
```

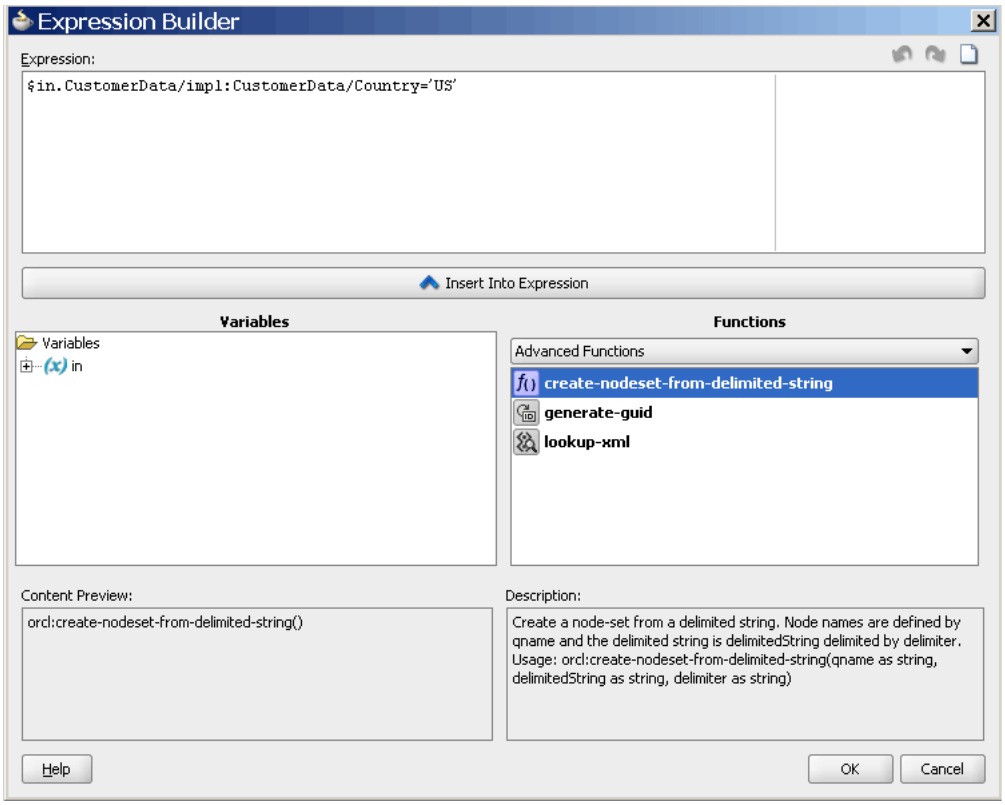
To specify an expression for filtering messages:

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:

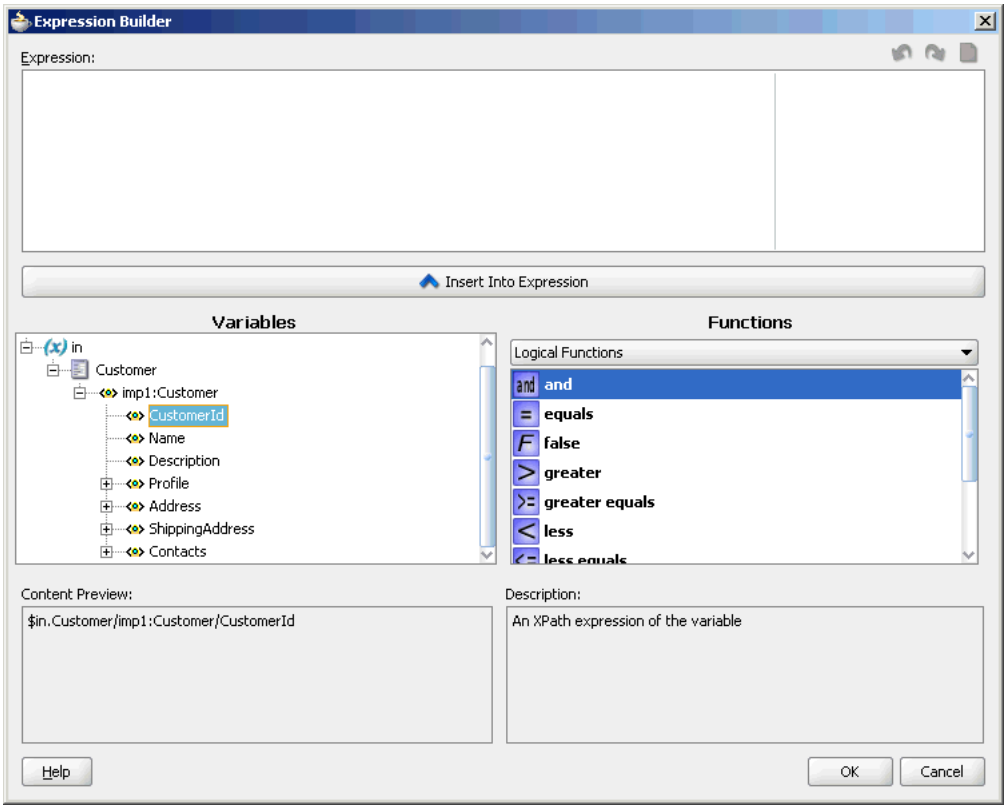
1. To the right of the **Filter Expression** field in the **Routing Rules** section, click the **Invoke Expression Builder** icon.

   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.

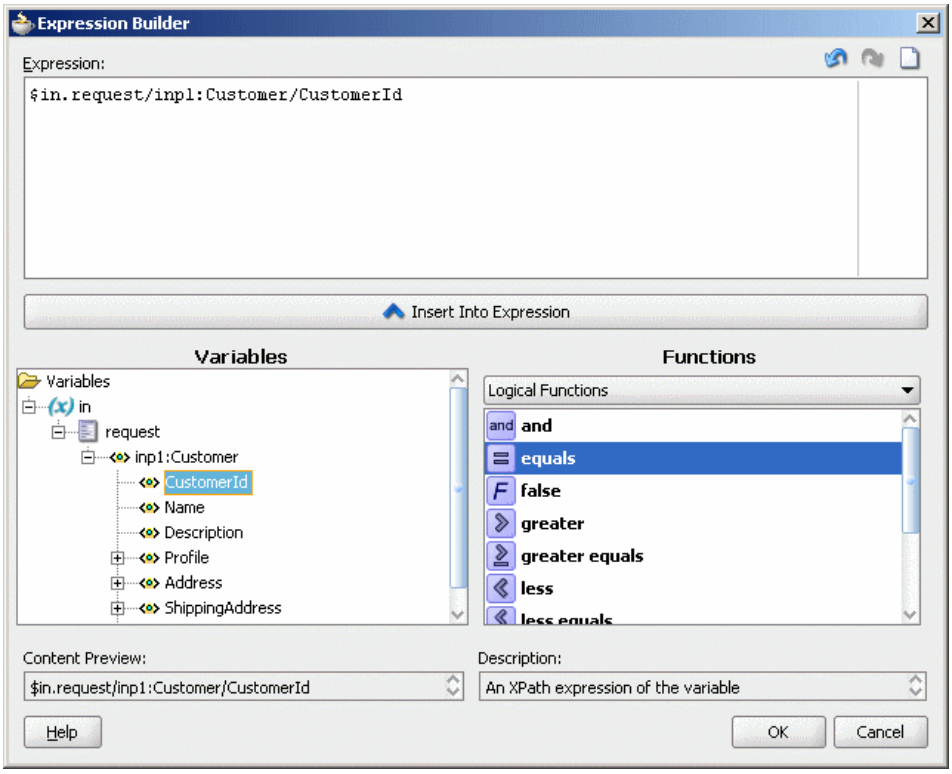
Figure 19–14 Expression Builder Dialog – Variables Element Selected



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.

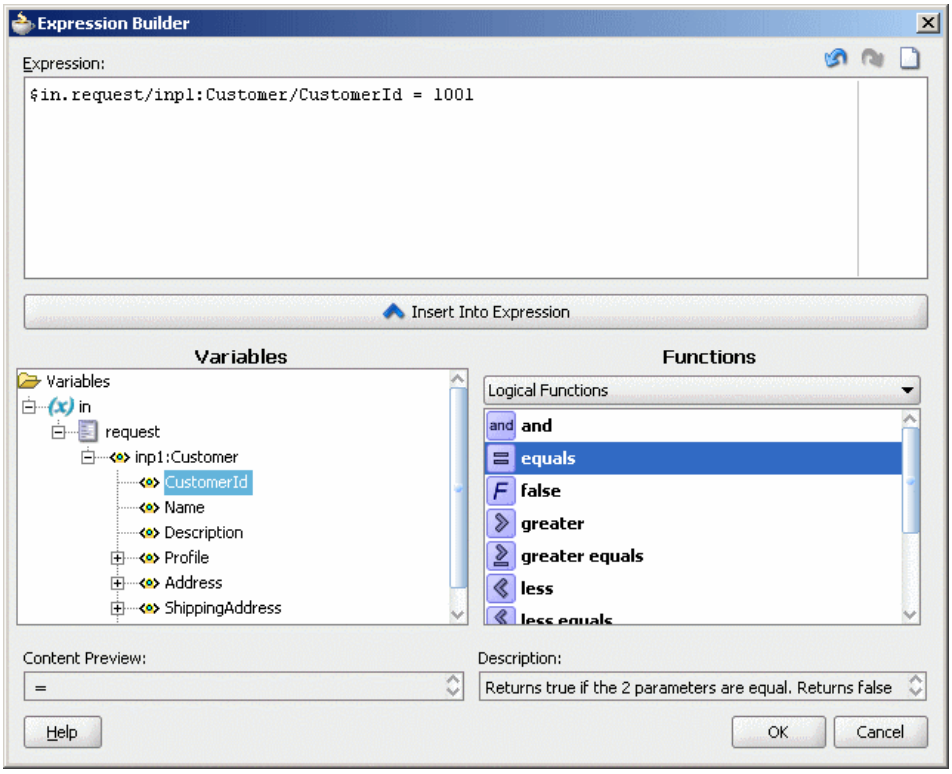
5. Click **Insert Into Expression**.

   The XPath expression for the selected function is inserted into the **Expression** field.

6. Complete the expression.

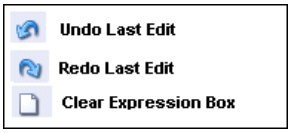
   In this example, the Customer ID must equal`1001` to evaluate to true, as shown in Figure 19–16.

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:

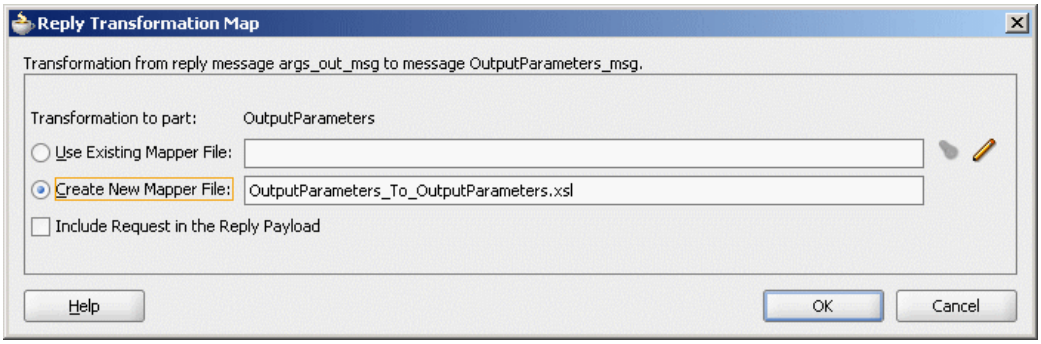
1. In the **Routing Rules** section, click the **Select an existing mapper file or create a new one** icon to the right of the **Transform Using** field.

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.

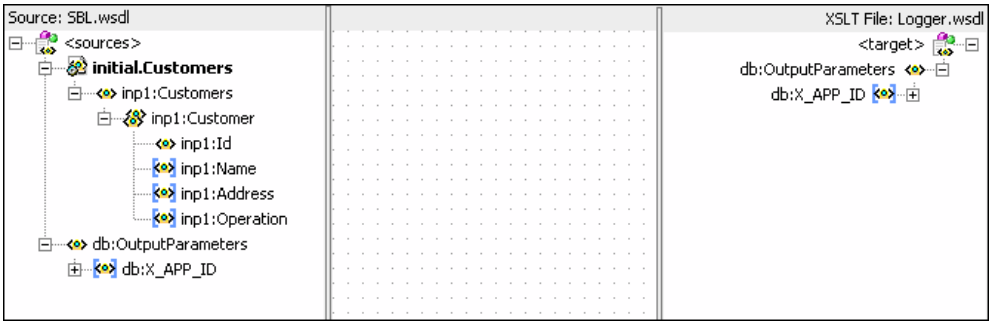
Figure 19–18 Reply Transformation Map Dialog



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."

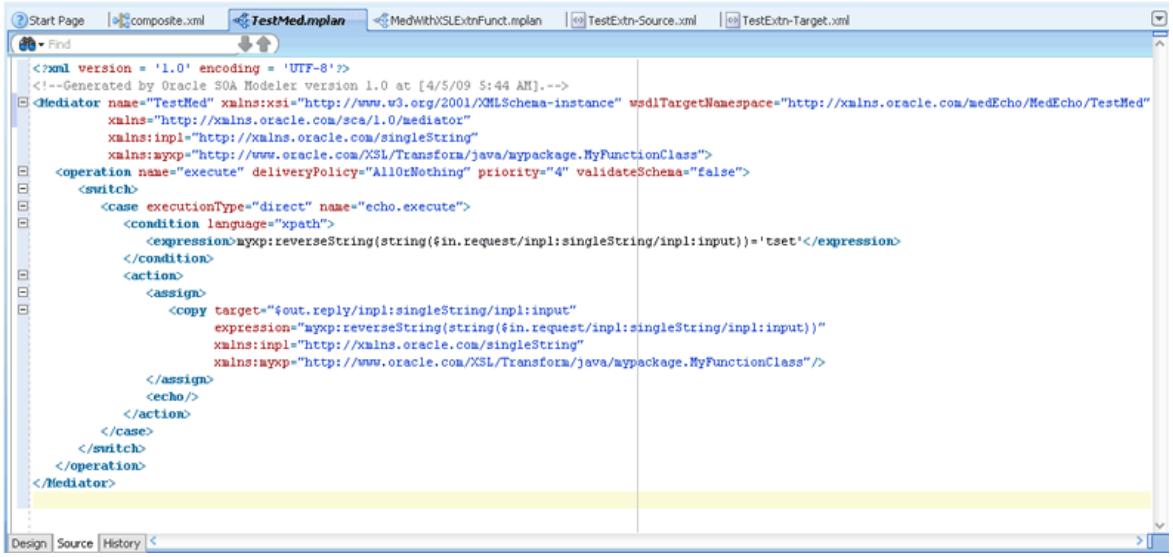
To add user-defined extension functions:

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

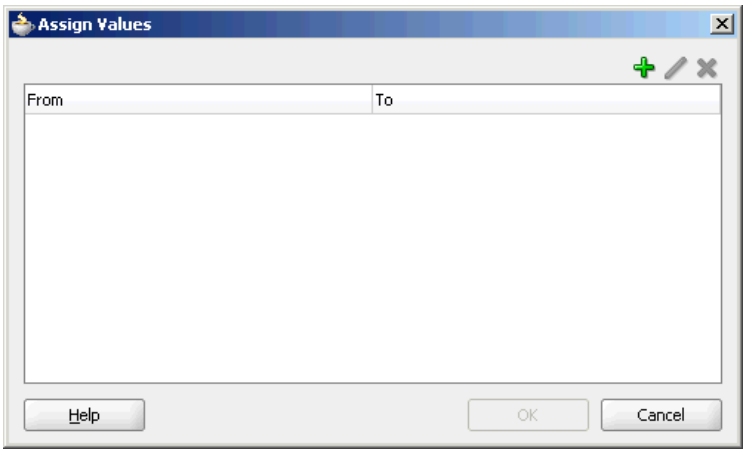


8. Invoke the test page with a suitable payload.

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

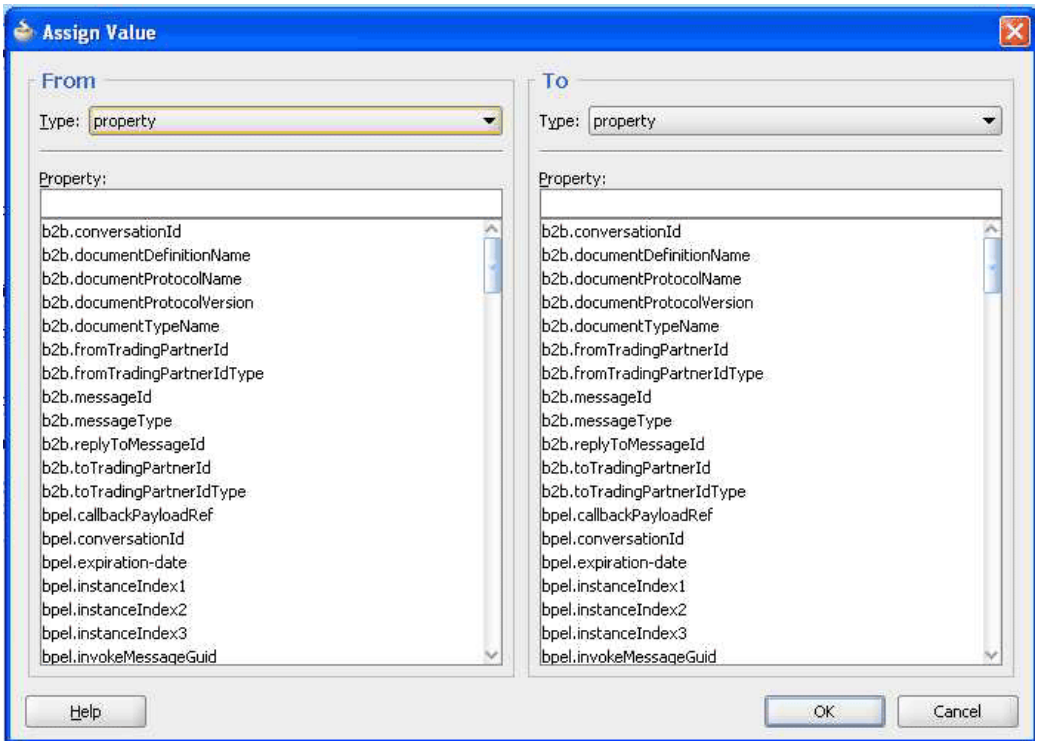


To set the properties of the target message:

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.

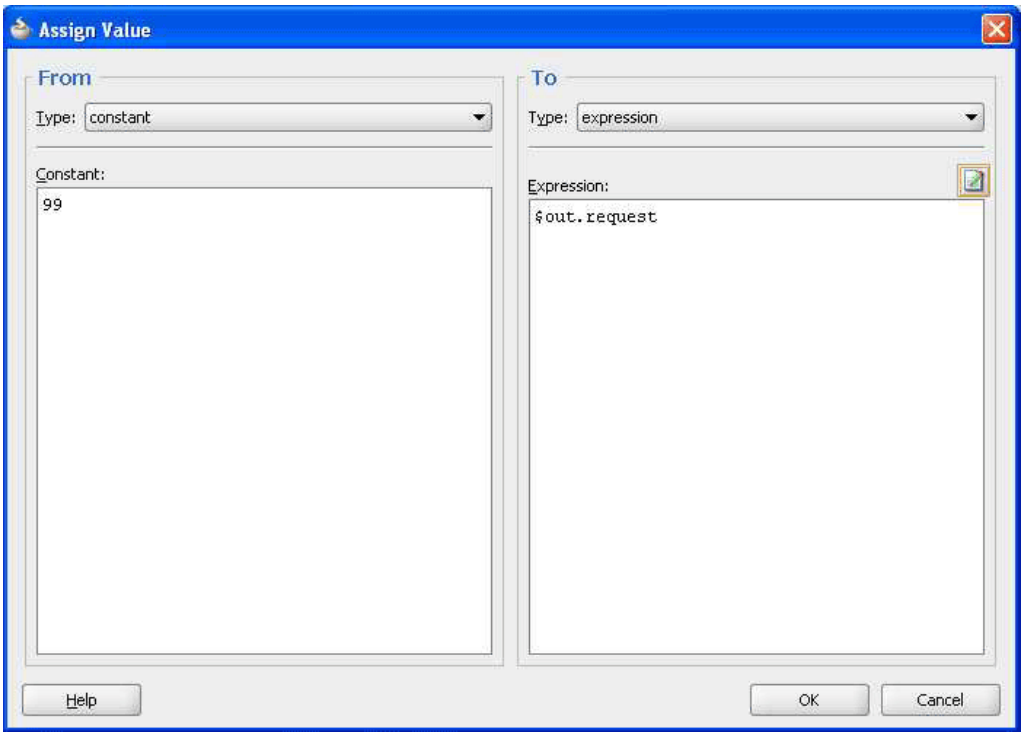
For more information about the Expression Builder dialog, see Section 19.2.2.7, "How to Specify an Expression for Filtering Messages."

- **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	`<copy expression="$in.property.jca.file.FileName" target="$out.property.jca.file.FileName"/>`
Constant	Property	`<copy value="ConstantNameAssigned.xml" target="$out.property.jca.file.FileName"/>`

Table 19–2 Possibilities on Payload

Source	Target	Example
XPath Expression	Property	`<copy expression="concat('ExprPropMed','-',oraext:generate-guid())" target="$out.property.jca.file.FileName" xmlns:oraext="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc"/>`
XPath Expression (below part level)	Property	`<copy expression="$in.body/imp1:request/ProductReq/Make" target="$out.property.jca.file.FileName" xmlns:imp1="http://xmlns.oracle.com/psft"/>`
Property	XPath Expression (below part level)	`<copy value="$in.property.jca.file.FileName" target="$out.request/inp1:request/ProductReq/Model" xmlns:inp1="http://xmlns.oracle.com/psft"/>`
Constant	XPath Expression (below part level)	`<copy value="ConstantModel" target="$out.request/inp1:request/ProductReq/Model" xmlns:inp1="http://xmlns.oracle.com/psft"/>`
XPath Expression	XPath Expression	`<copy expression="$in.body" target="$out.request"/>`

Table 19–2 (Cont.) Possibilities on Payload

Source	Target	Example
XPath Expression (below part level)	XPath Expression (below part level)	`<copy expression="$in.body/imp1:request/ProductReq/Make" target="$out.request/imp1:request/ProductReq/Model" xmlns:imp1="http://xmlns.oracle.com/psft"/>`

Table 19–3 Possibilities on Header

Source	Target	Example
XPath Expression (below part level)	Property	`<copy expression="$in.header.inp1_header/inp1:header/Name" target="$out.property.jca.file.FileName" xmlns:inp1="http://xmlns.oracle.com/psft"/>`
Property	XPath Expression (below part level)	`<copy value="$in.property.jca.file.FileName" target="$out.header.inp1_header/inp1:header/Name" xmlns:inp1="http://xmlns.oracle.com/psft"/>`
Constant	XPath Expression (below part level)	`<copy value="NewID.xml" target="$out.header.inp1_header/inp1:header/Id" xmlns:inp1="http://xmlns.oracle.com/psft"/>`
Constant	XPath Expression (below part level)	`<copy value="sampleusername" xmlns:wsse1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" target="$out.header.wsse1_Security/wsse1:Security/wsse1:UsernameToken/wsse1:Username"/>`
XPath Expression	XPath Expression	`<copy target="$out.header.inp1_header" expression="$in.header.inp1_header" xmlns:inp1="http://xmlns.oracle.com/psft"/>`
XPath Expression (below part level)	XPath Expression (below part level)	`<copy target="$out.header.inp1_header/inp1:header/Name" expression="$in.header.inp1_header/inp1:header/Id" xmlns:inp1="http://xmlns.oracle.com/psft"/>`

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/imp1:request/ProductReq/Make"
 xmlns:imp1="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/inp1:request/ProductReq/Make" value="NewMakeValue"
 xmlns:inp1="http://xmlns.oracle.com/psft"/>
```

In this example, `$out.request/inp1: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="$out.header.inp1_header" expression="$in.header.inp1_header"
  xmlns:inp1="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/inp1:request/ProductReq/Model"
 xmlns:inp1="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/inp1:request/ProductReq/Model"
 xmlns:inp1="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="$out.request/imp1:ProductReq"
 expression="$in.body/imp1:request/ProductReq"
 xmlns:imp1="http://xmlns.oracle.com/psft"/>
```

In this case, the source element evaluated from `$in.body/imp1: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="$out.request/imp1:request/ProductReq/Make"
 expression="$in.body/imp1:request/ProductReq/Model"
xmlns:imp1="http://xmlns.oracle.com/psft"/>

<copy target="$out.request/imp1:request/ProductReq/Make"
 expression="$in.body/imp1:request/Description"
xmlns:imp1="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

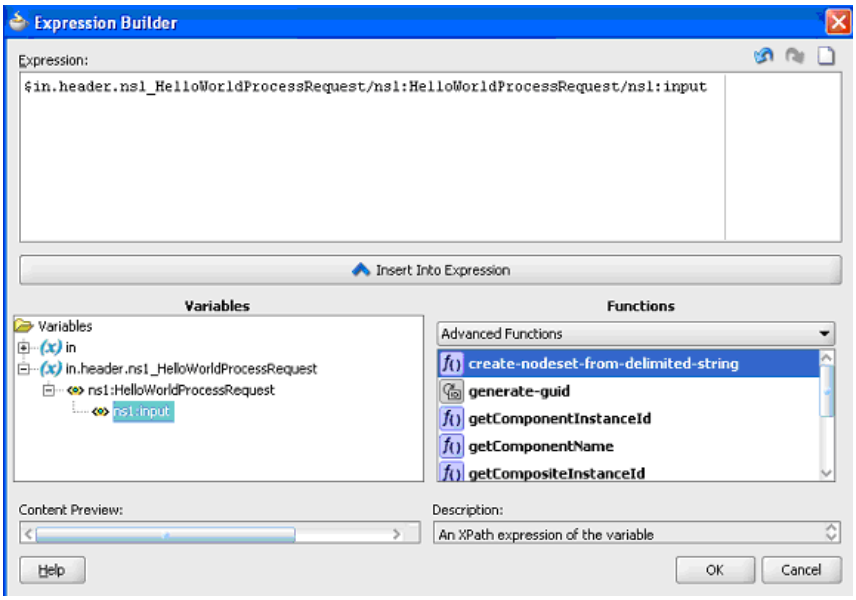
```
$in.header.{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
ext-1.0.xsd}_Security/
{"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xs
d"}:
Security/{"http://www.globalcompany.com/ns/OrderBooking"}:Foo/Priority
```

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.

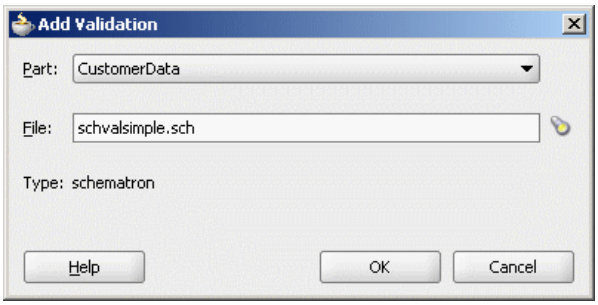
5. Select a Schematron file and click **OK**.

   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.

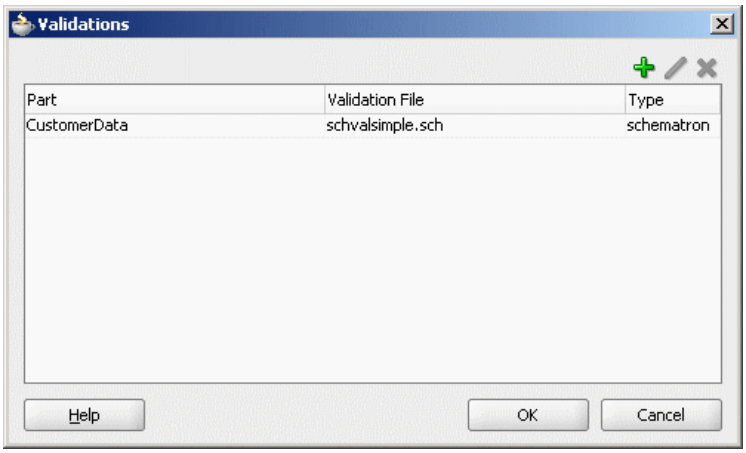
Figure 19–25 Add Validation Dialog



6. Click **OK**.

   The Validation dialog is updated, as shown in Figure 19–26.

Figure 19–26 Validation Dialog



7. 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

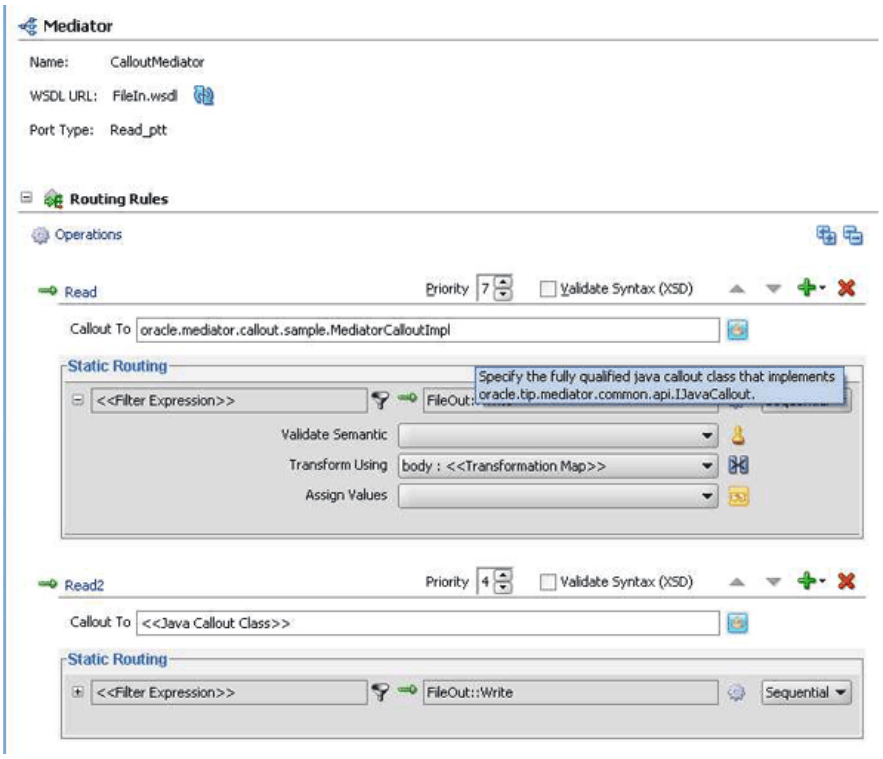
   `http://www.schematron.com`

   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.

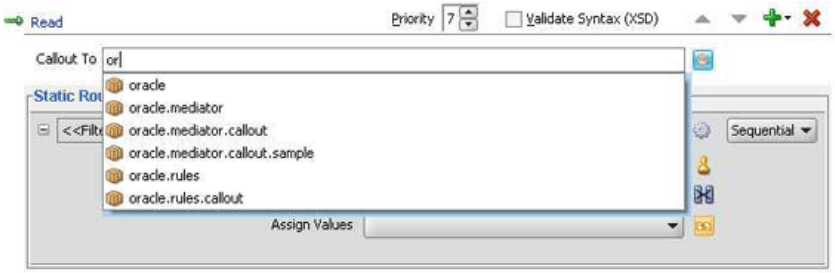
If you want to make the Java callout class available to multiple Oracle Mediators, copy the JAR file containing the Java class to the `$DOMAIN_HOME/lib` folder.

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.

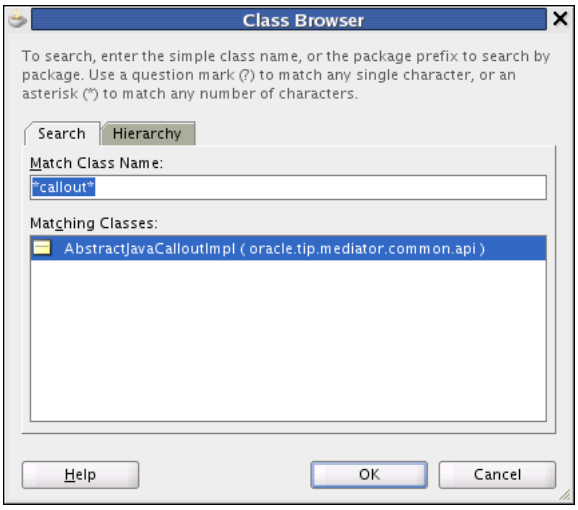
Figure 19–28 Callout To Field



- 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

```
changexmldoc = XmlUtils.getXmlDocument(ChangedDoc);
String mykey = "request";
message.addPayload(mykey,changexmldoc.getDocumentElement());
```

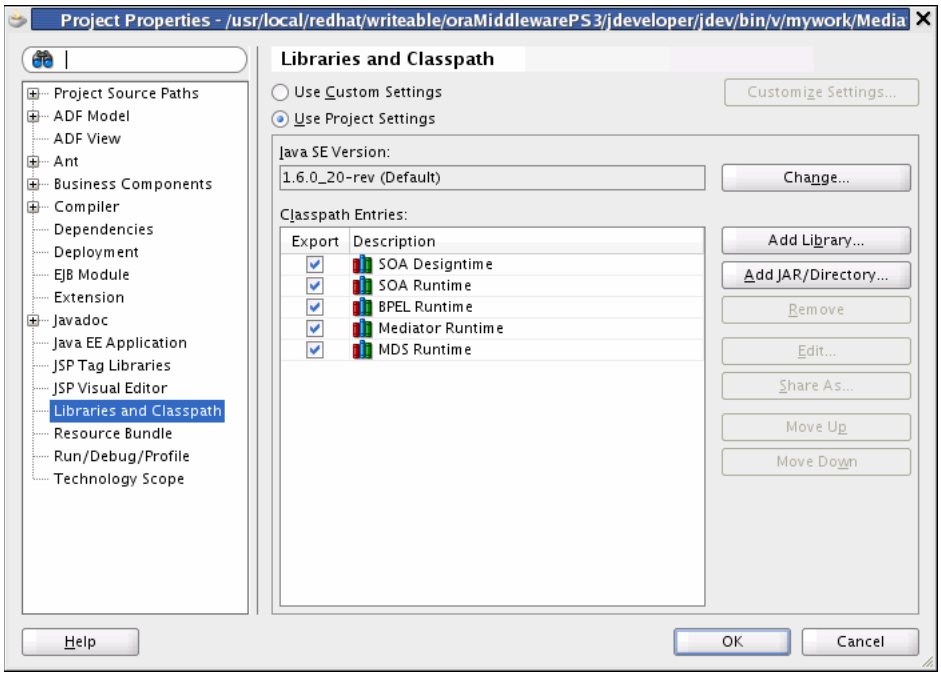
To enable domain value map and cross reference functions:

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.

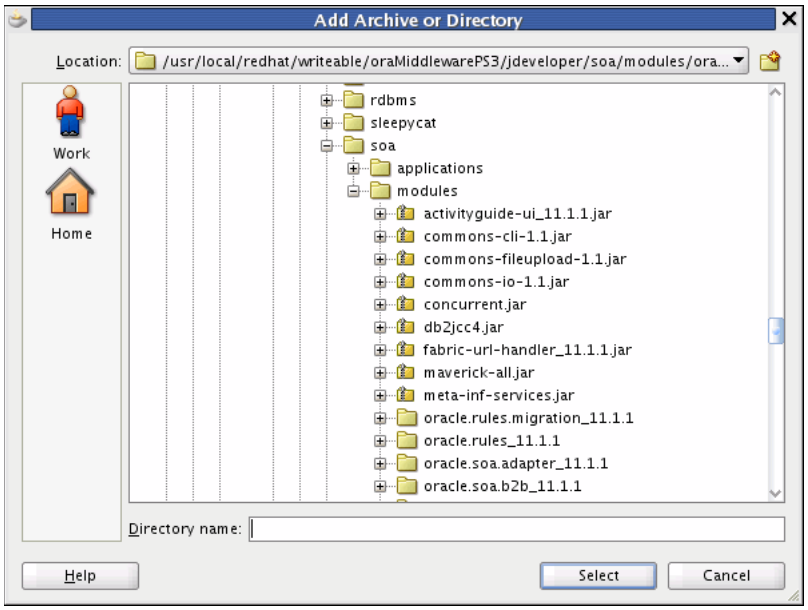
Figure 19–30 Libraries and Classes on the Project Properties Dialog



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
`initialize`	This method is invoked when the callout implementation class is instantiated for the first time.
`preRouting`	This method is called before Oracle Mediator starts executing the cases. You can customize this method to include validations and enhancements.
`preRoutingRule`	This method is called before Oracle Mediator starts executing any particular case. You can customize this method to include case-specific validations and enhancements.
`preCallbackRouting`	This method is called before Oracle Mediator finishes executing callback handling. You can customize this method to perform callback auditing and custom fault tracking.
`postRouting`	This method is called after Oracle Mediator finishes executing the cases. You can customize this method to perform response auditing and custom fault tracking.
`postRoutingRule`	This method is called after Oracle Mediator starts executing the cases. You can customize this method to perform response auditing and custom fault tracking.
`postCallbackRouting`	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
`addPayload`	This method sets a payload of the Oracle Mediator messages.
`addProperty`	This method adds a property to the Oracle Mediator messages.
`addHeader`	This method adds a header to the Oracle Mediator messages.
`getProperty`	This method retrieves Oracle Mediator message properties by providing the property name.
`getProperties`	This method retrieves Oracle Mediator message properties.
`getId`	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.
`getPayload`	This method retrieves a payload of the Oracle Mediator messages.
`getHeaders`	This method retrieves a header of the Oracle Mediator messages.
`getComponentDN`	This method retrieves a componentDN for the Oracle Mediator service component.

Notes:

- The `oracle.tip.mediator.common.api.AbstractJavaCalloutImpl` class is a dummy implementation[1] 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.

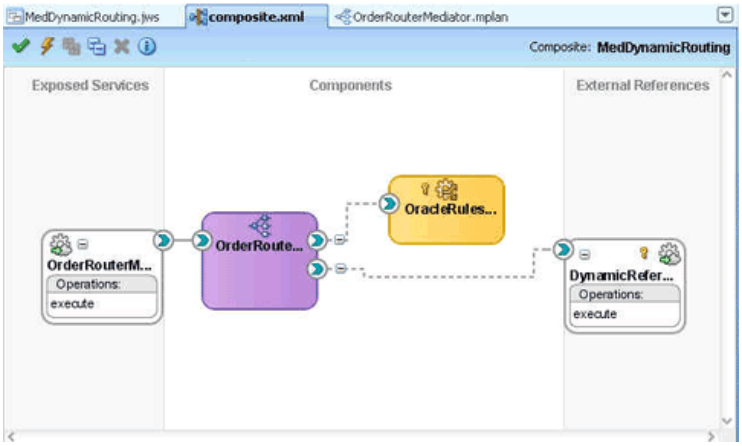
How to create dynamic routing rules:

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)

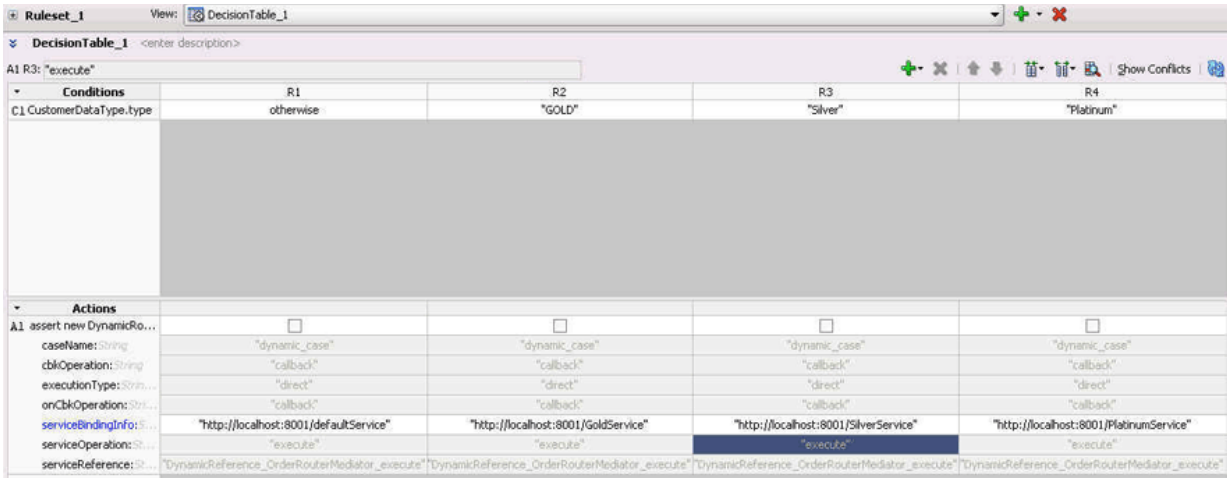
  From a rule engine perspective, a decision table is a collection of rules with the same fact type model elements in the condition and action part of the rules. The decision table enables you to visualize rules in a tabular format. As part of creating the business rule service component, a new decision 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



2. 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

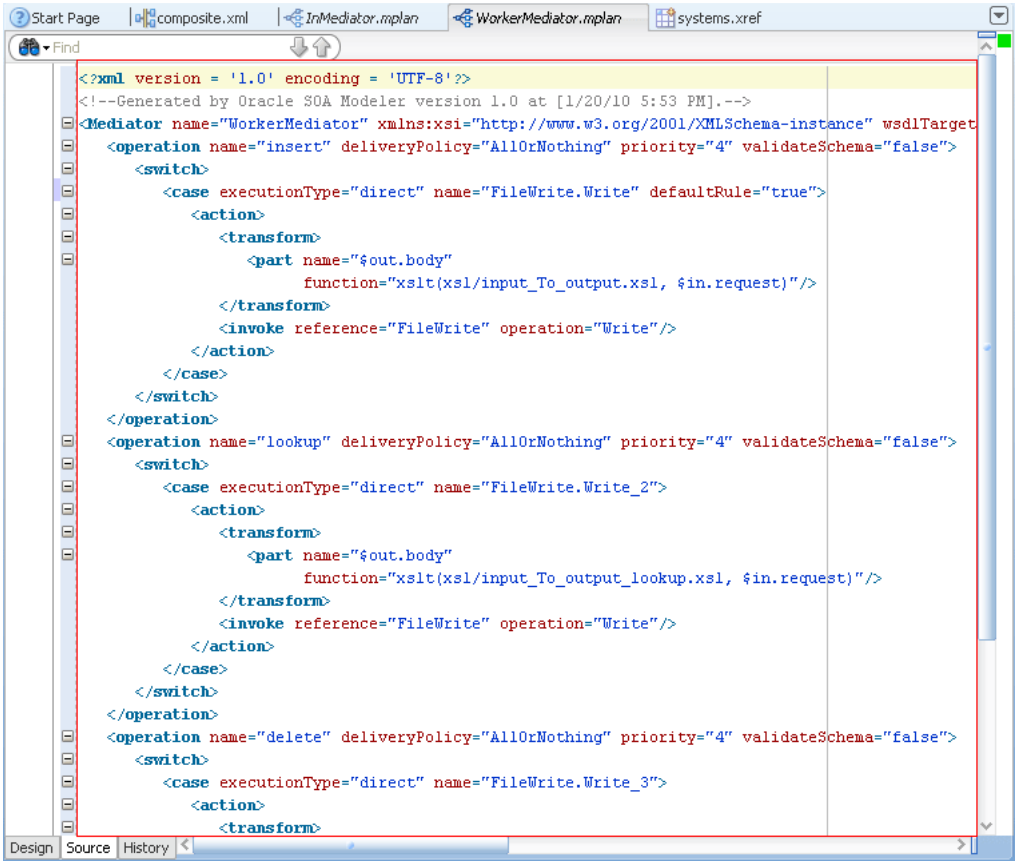
The current behavior of a pre-Java callout or post-Java callout works in the same way as for other routing rules. For the purpose of Java callouts, the default routing rule is considered another routing rule. Therefore, for the scenarios in which the default routing rule is executed, the `postRouting()` 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.

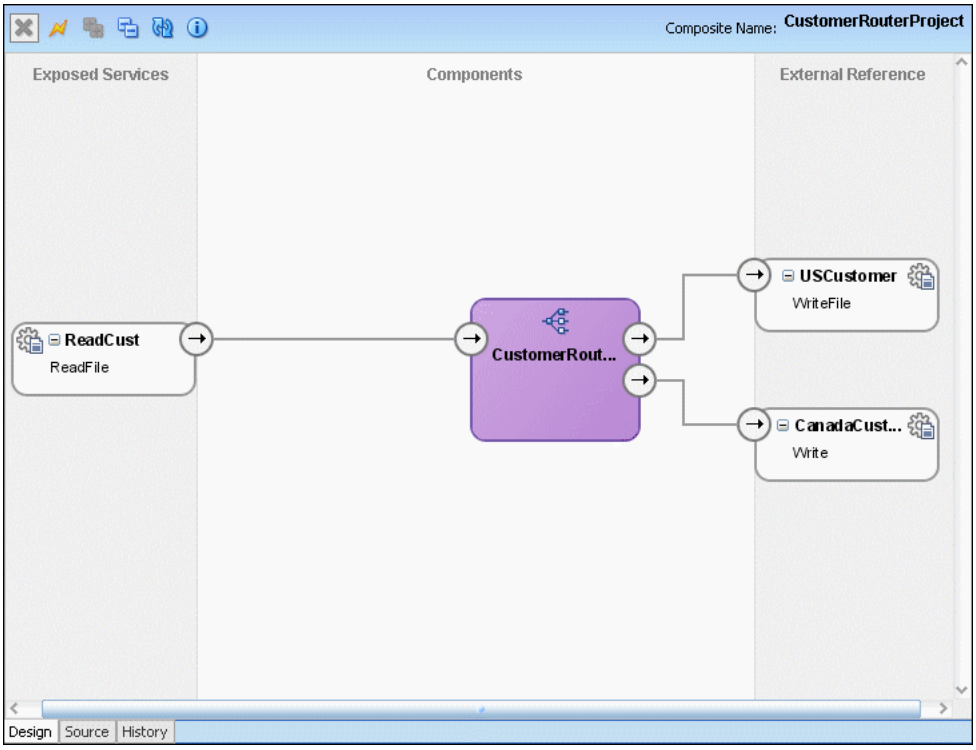
The CustomerRouter use case consists of the following steps:

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.**

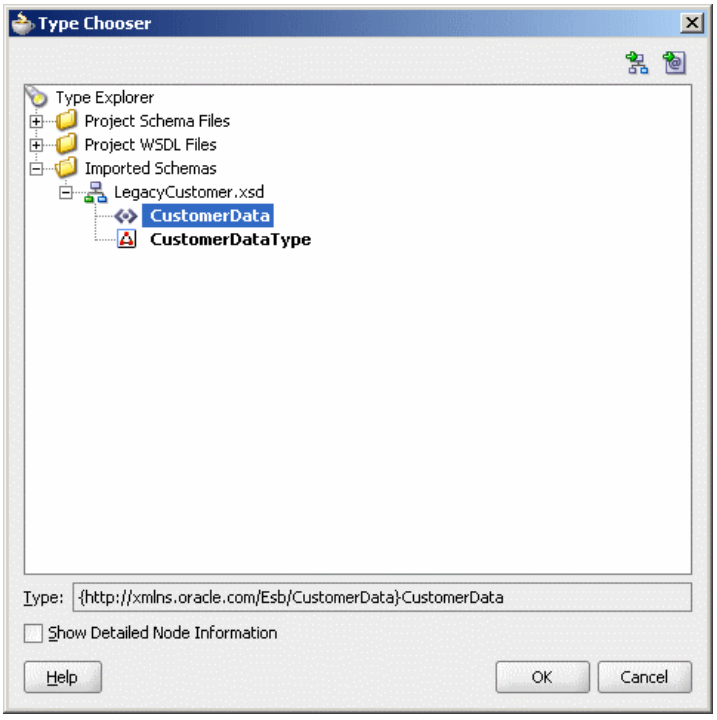
    The File Polling page is displayed.
13. Change the **Polling Frequency** field value to **10 seconds,** and then click **Next.**

    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.

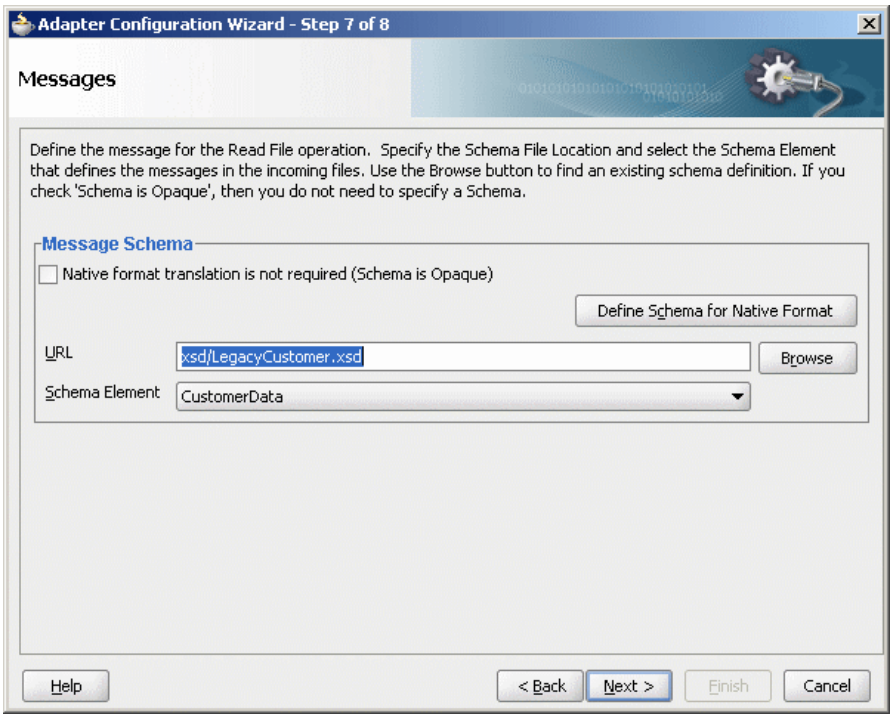
Figure 19–38 Type Chooser - CustomerData



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**.

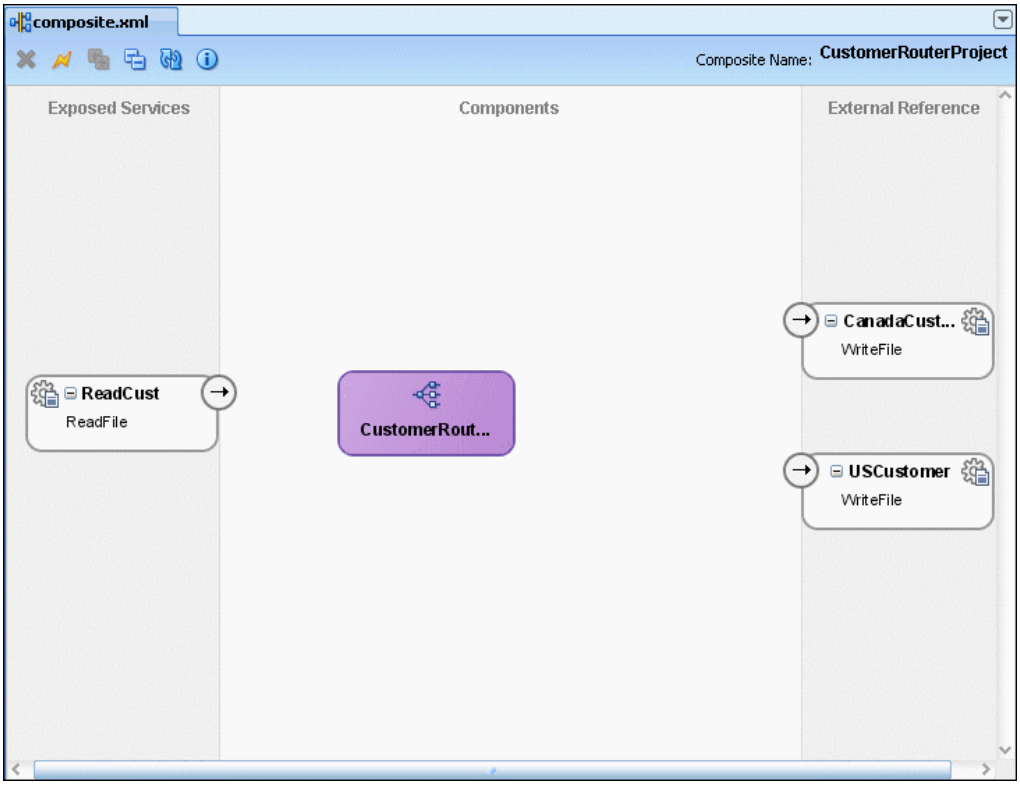
    The Import Schema File dialog is displayed.
15. To the right of the **URL** field, click **Search** and select the **USCustomer.xsd** file present in the **Samples** folder.
16. Click **OK**.
17. Expand the navigation tree to **Type Explorer\Imported Schemas\USCustomer.xsd**, and then select **Customer**.
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

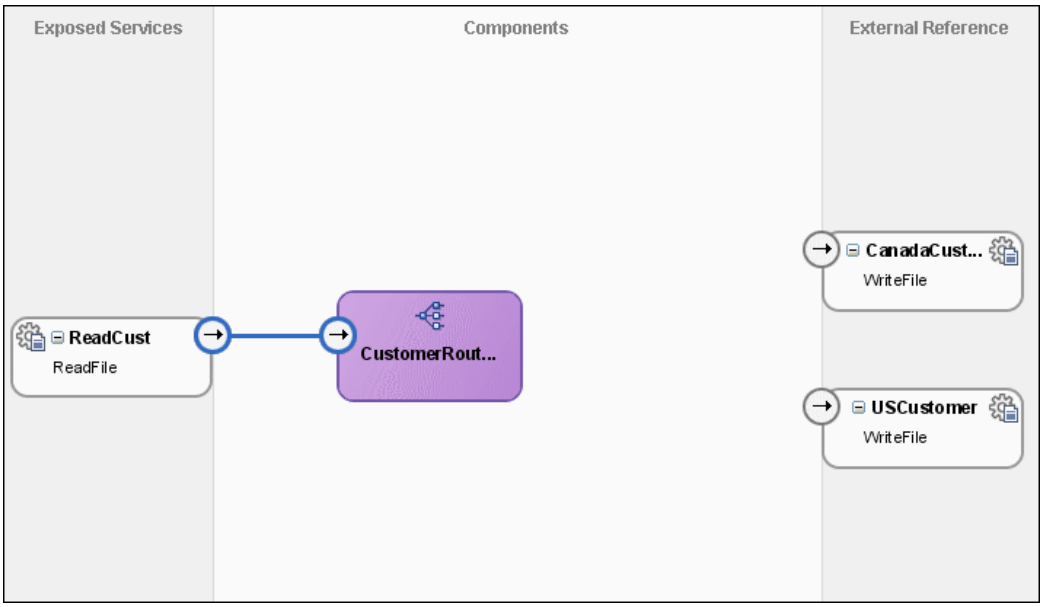
You must specify the path that messages take from the ReadCust adapter service to external references.

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

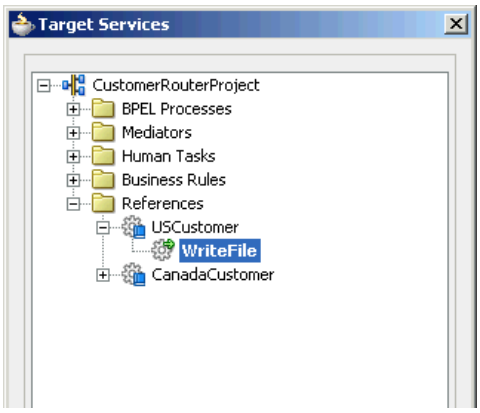


2. Double-click the **CustomerRouter** Oracle Mediator in the Mediator Editor.
3. In the **Routing Rules** section, click **Add** to the extreme right side of **ReadFile**, and then click **static routing rule**.

   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**.

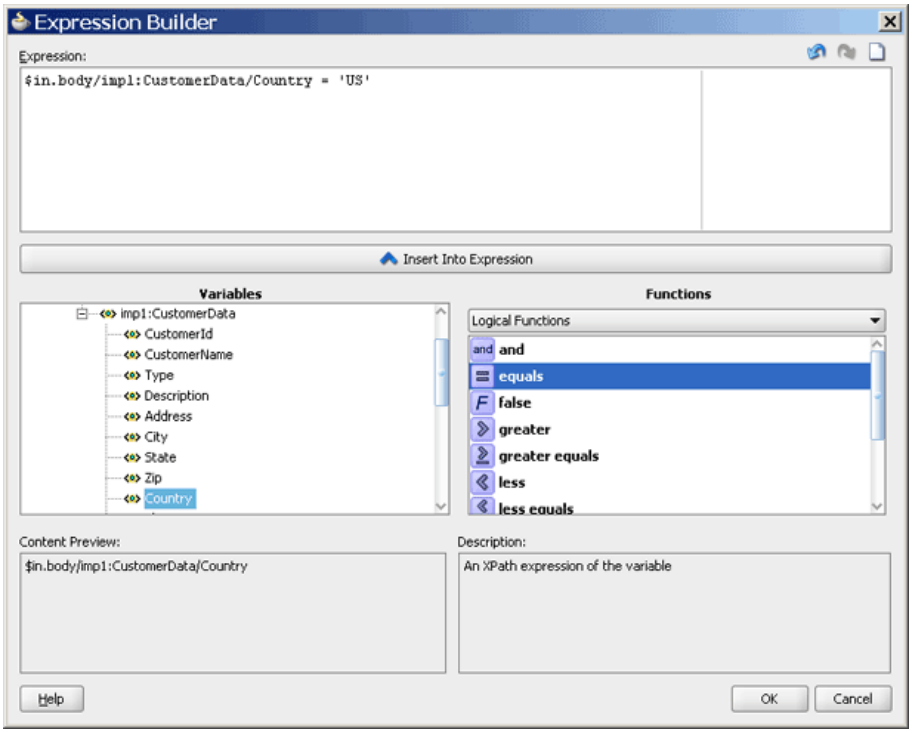
   The **Routing Rules** section is displayed.
7. Next to the **<<Filter Expression>>** field, click the **filter** icon to create a filter expression for this routing rule.

   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'
    ```

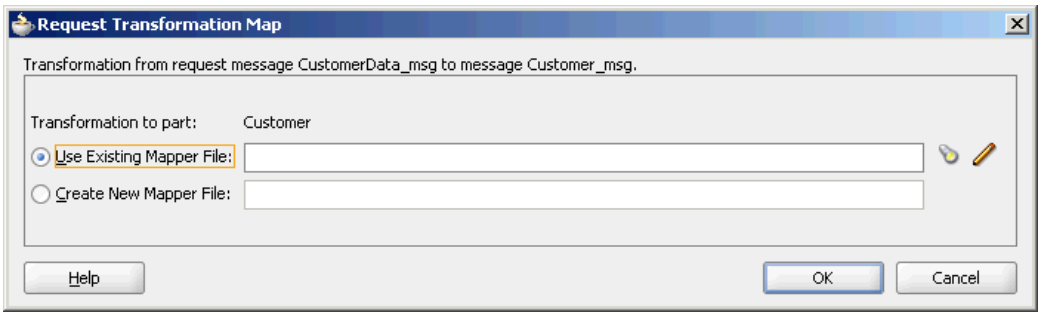
11. Click **OK**.

    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

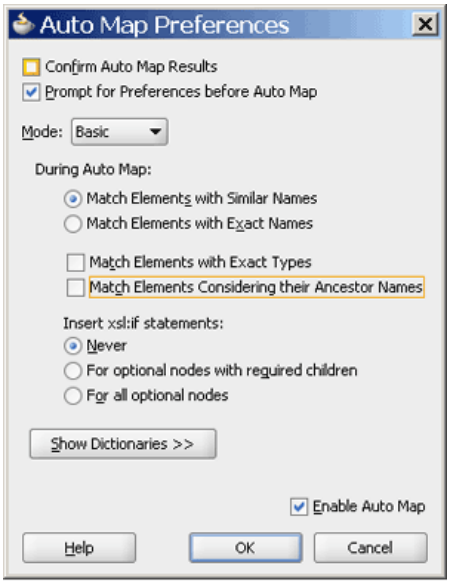
14. Drag and drop the **imp1:CustomerData** source element to the **imp1:Customer** target element.

    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.

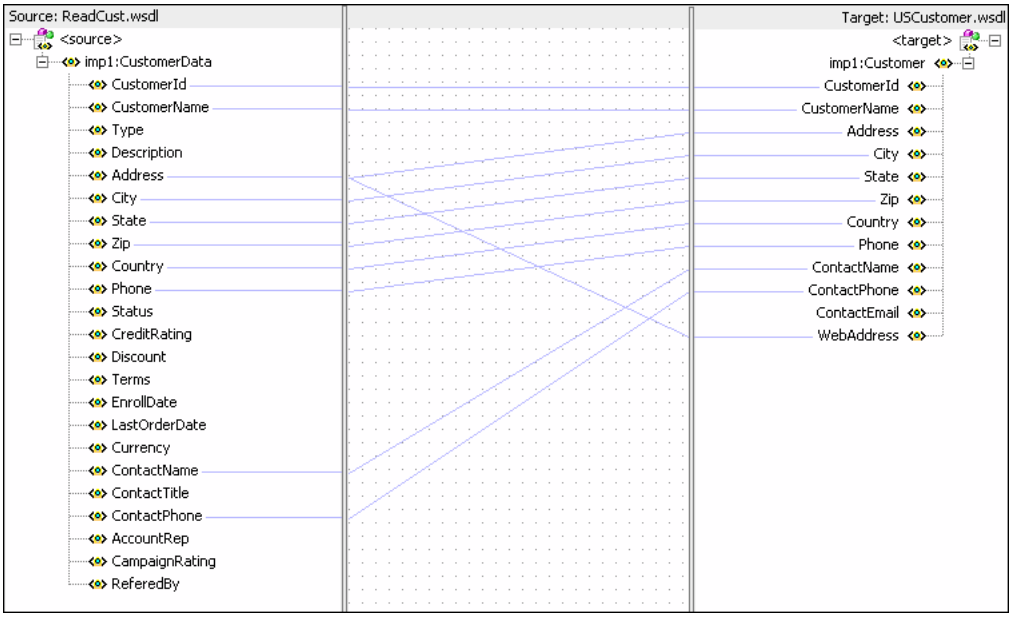
Figure 19–46 Auto Map Preferences Dialog



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/imp1: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.

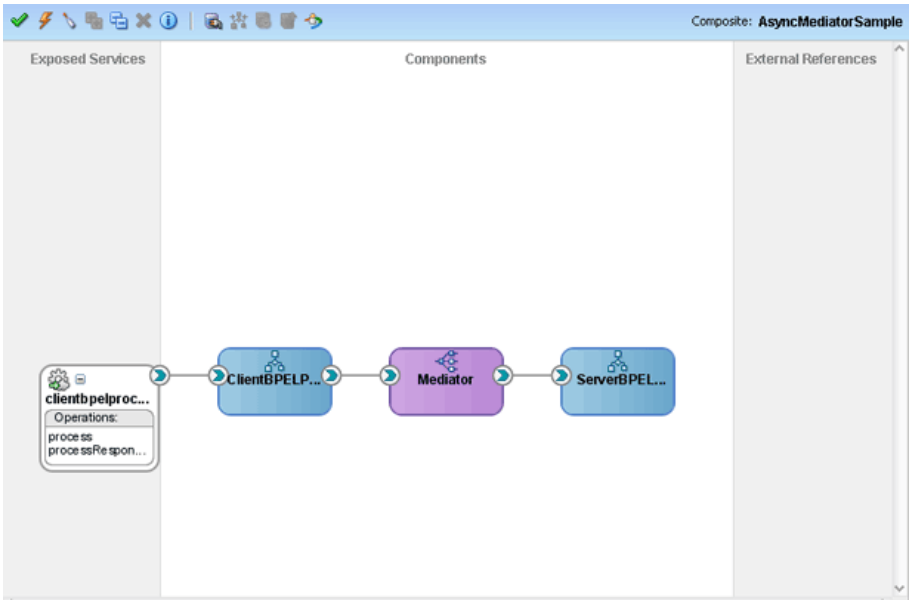
For detailed information about these steps, see Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper."

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.

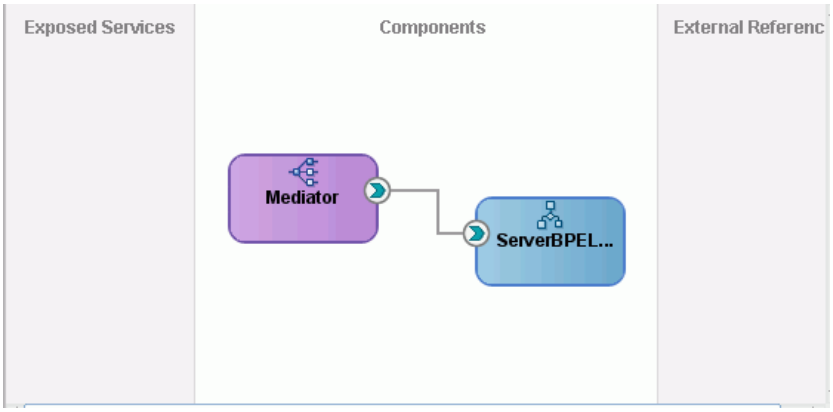
   The Create Mediator dialog is displayed.

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



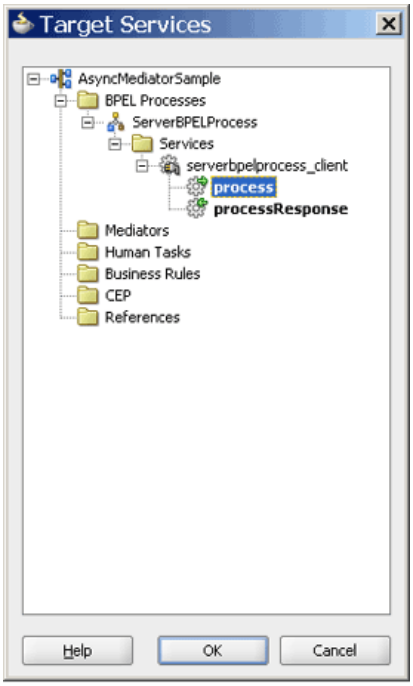
7. Double-click the **Mediator** Oracle Mediator.

   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.

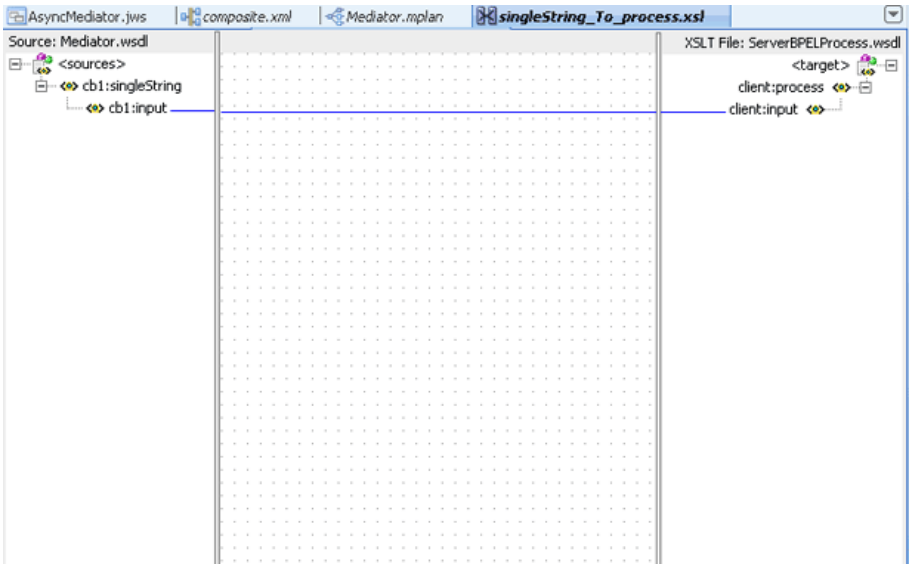
    The Request Transformation Map dialog is displayed.
13. Select **Create New Mapper File** and click **OK**.

    The XSLT Mapper is displayed and a target file named **singleString_To_process.xsl** 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.

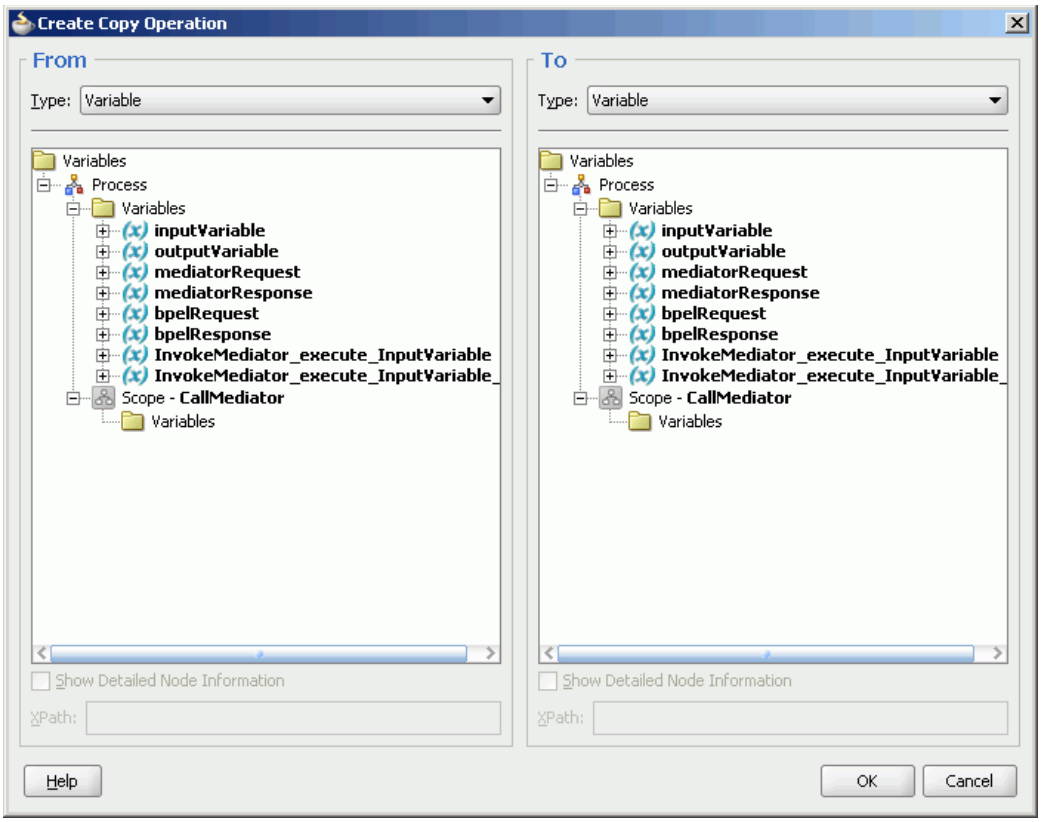
To create the receive activity:

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:

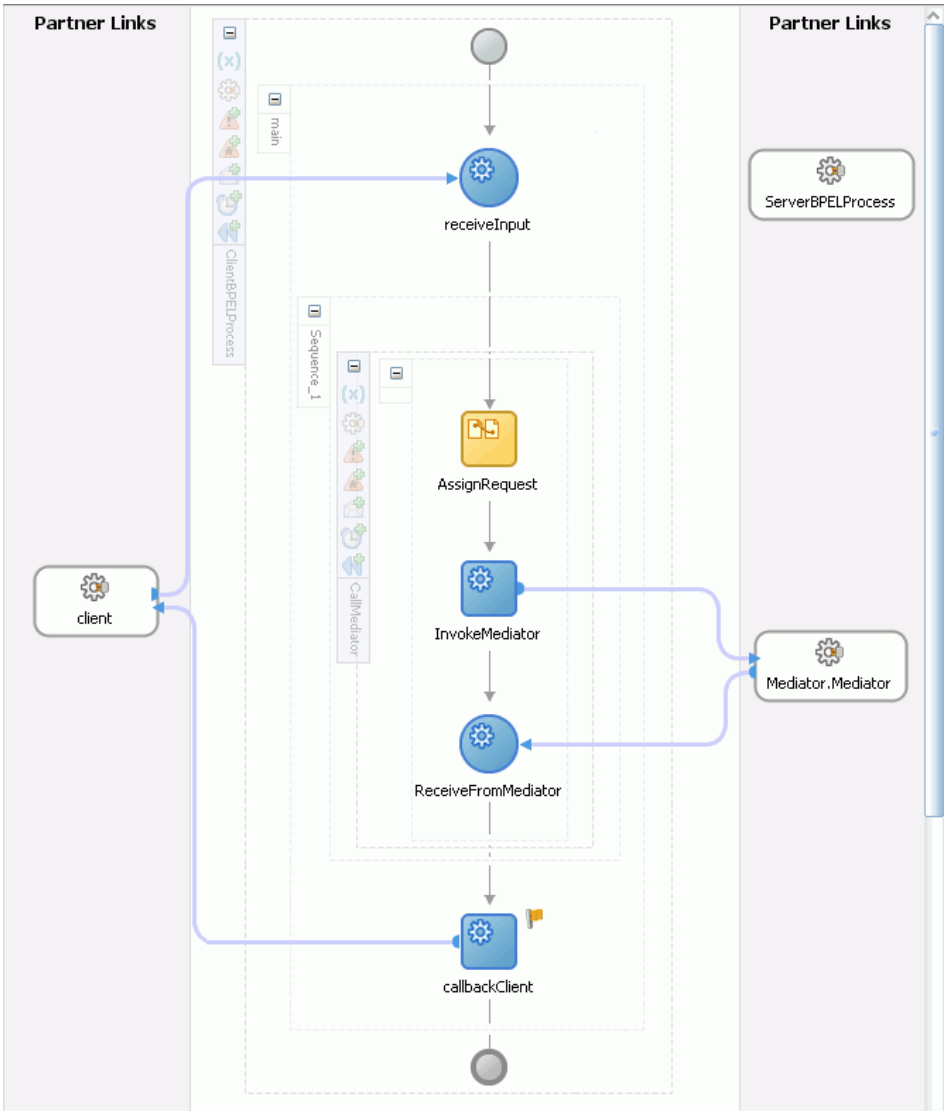
1. Drag and drop an **Assign** activity from the Component Palette between the **ReceiveFromMediator** and **InvokeMediator** activities in the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. In the **Name** field, enter `AssignRequest`.
4. Click the **Copy Operation** 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

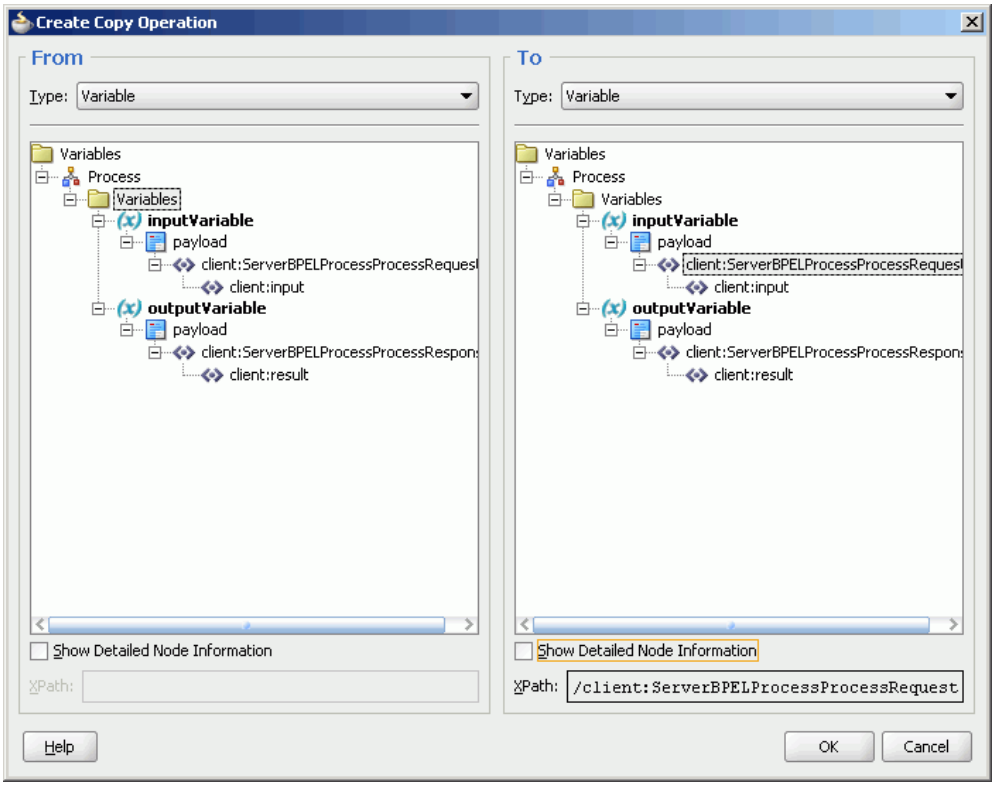


9. From the **File** menu, select **Save All**.

To create an assign activity in the ServerBPELProcess

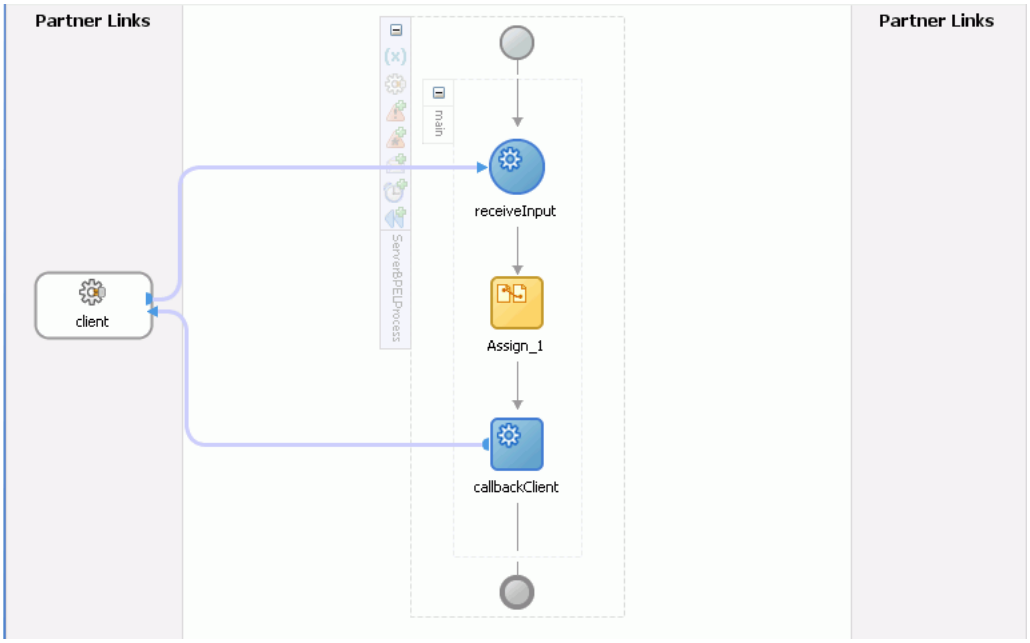
1. Double-click the **ServerBPELProcess** BPEL process. The Oracle BPEL Designer is displayed.
2. Drag and drop an **Assign** activity from the Component Palette between the **receiveInput** and **callbackClient** activities in the design area.
3. Double-click the **Assign** activity. The Assign dialog 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."

20

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"

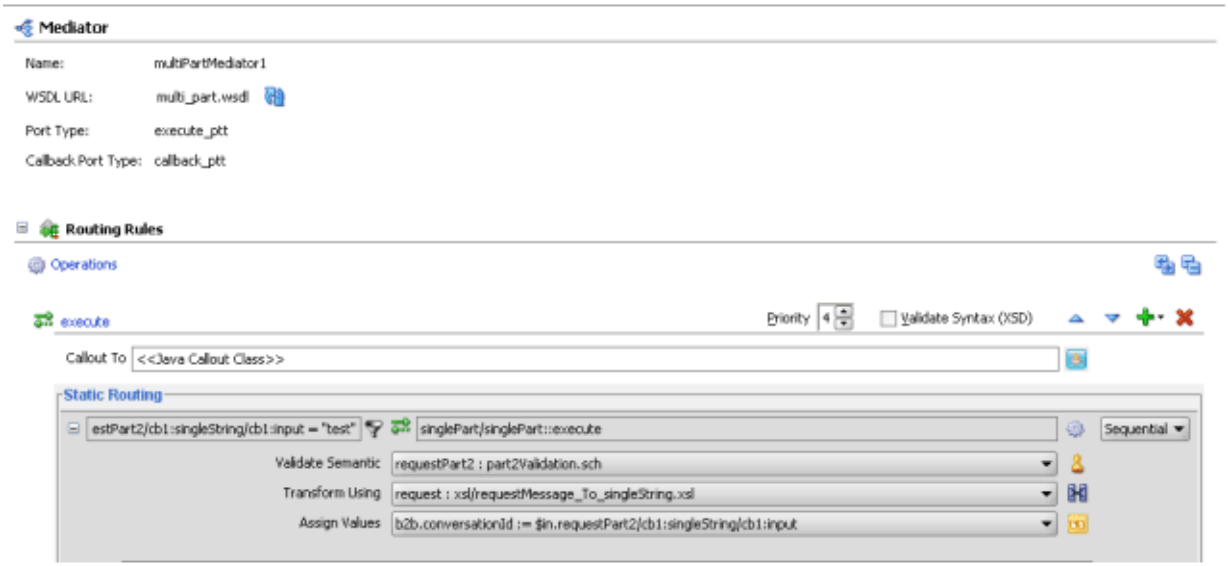
20.1 Introduction to Oracle Mediator Multipart Message Support

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.

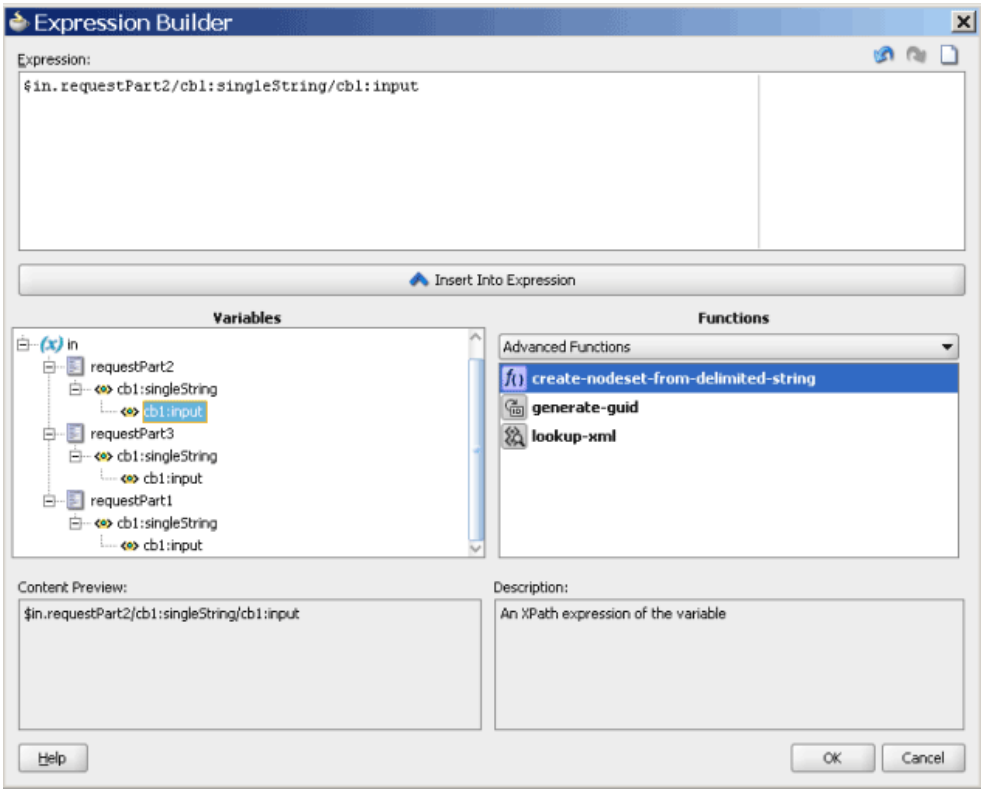
20.2.1 How to Work with Multipart Request Messages

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:

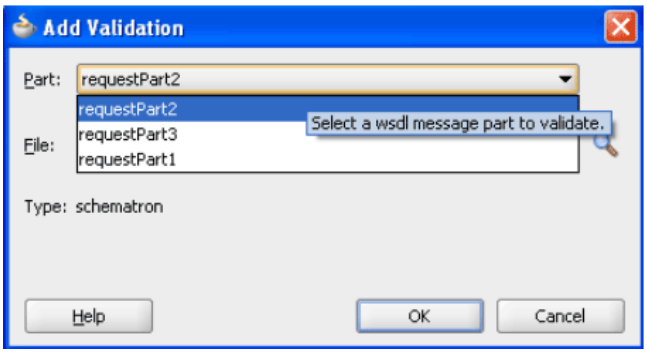
Figure 20–2 Expression Builder for a Multipart Request Source



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.

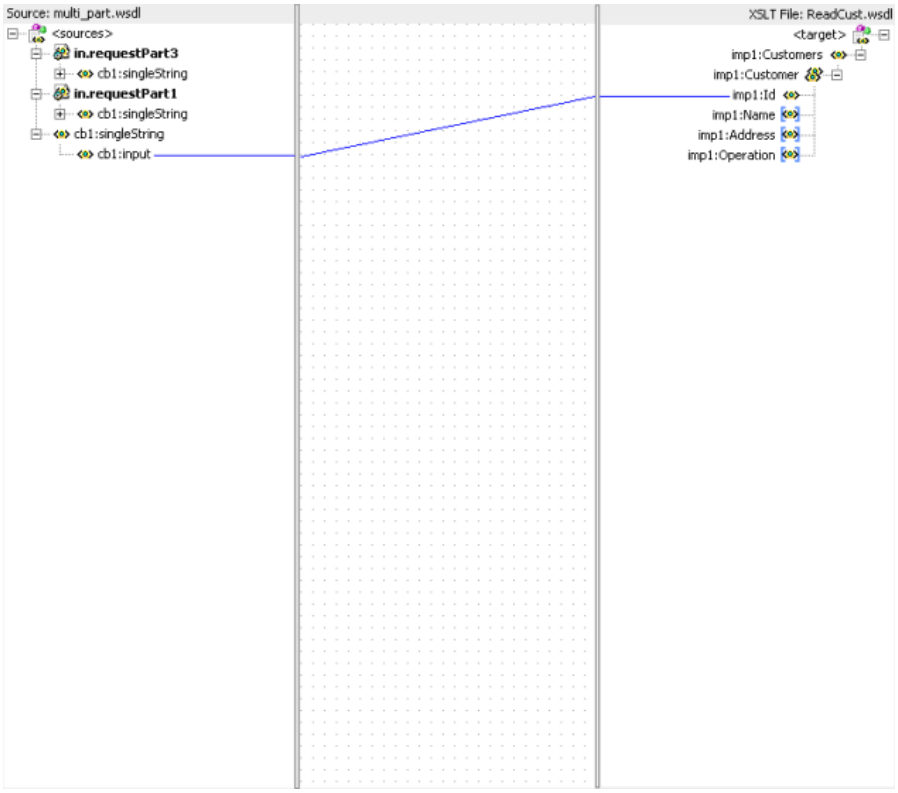
Figure 20–3 Add Validation Dialog for a Multipart Request Source



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

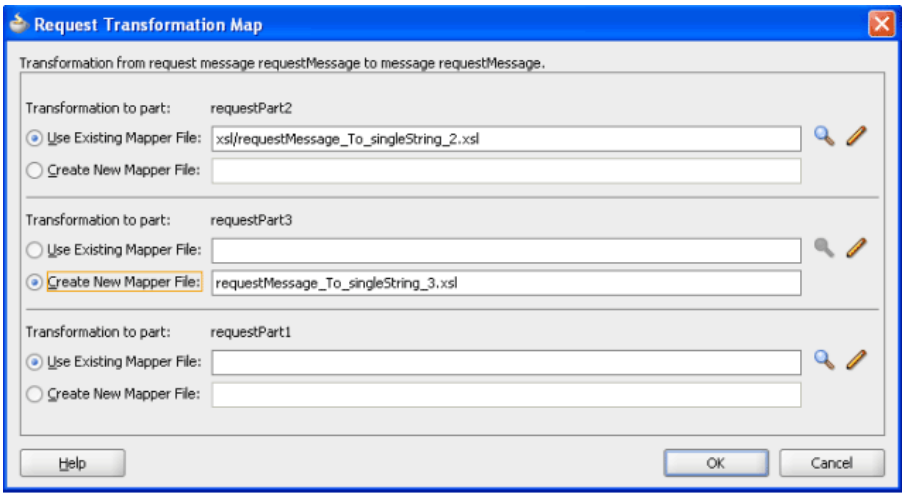
The method to create transformations and assign values to multipart reply, fault, and callback source messages is the same as working with request source messages.

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



21

Using Oracle Mediator Error Handling

This chapter describes how to handle errors with Oracle Mediator.

This chapter includes the following sections:

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 *N* 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 *N* 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 x 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"/>
```

```
        <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 (`.mplan` file).
  - `TYPE_METADATA_COMMON`: Contains other errors that occur during the handling of metadata.
- `TYPE_FATAL`: Contains fatal errors that are not easily recoverable.
  - `TYPE_FATAL_DB`: Contains database-related fatal errors, such as 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 enqueuing 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: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>
```

22

Resequencing in Oracle Mediator

This chapter describes support for message resequencing in Oracle Mediator. It contains the following sections:

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 ones 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>
        <ReferedBy>Housten America Taxco</ReferedBy>
</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 period[1], 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 ha 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), msg09(06), msg05(07), msg08(08), msg12(09), msg11(10), msg10(12), msg07(13)
msg02(05)/2:00:20	
msg03(01)/2:00:30	
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.

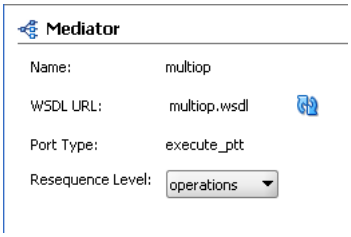
22.3.1 How to Specify the Resequencing Level

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

The options in the **Resequence Options** section change depending on the resequencing mode you select.

22.3.2 How to Configure the Resequencing Strategy

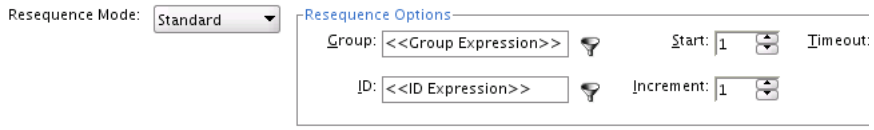
This section provides instructions on how to configure the three different types of resequencing strategies.

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



3. Fill in the fields listed inTable 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 `groupIDExpression`.	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 `sequenceIDExpression`.	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 `timeOutDuration`.	0[1]	N
Start	The starting number of the ID sequence. If you are editing the MPLAN file directly, the corresponding element is named `sequenceStart`.	1	N
Increment	The increment of the ID sequence. If you are editing the MPLAN file directly, the corresponding element is named `sequenceIncrement`.	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

3. In the **Group** field, enter the XPath expression pointing to the field in the incoming message on which grouping is performed.

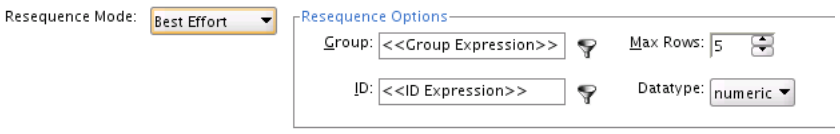
Notes: If you are modifying the `MPLAN` file directly, the name of the XML element is `groupIDExpression`. There is no default value, and the field is not mandatory.

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 `groupIDExpression`.	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 `sequenceIDExpression`.	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 `sequenceIDDataType`.	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 `timeWindow`.	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 `maxRowsRetrieved`.	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.

23

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:

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

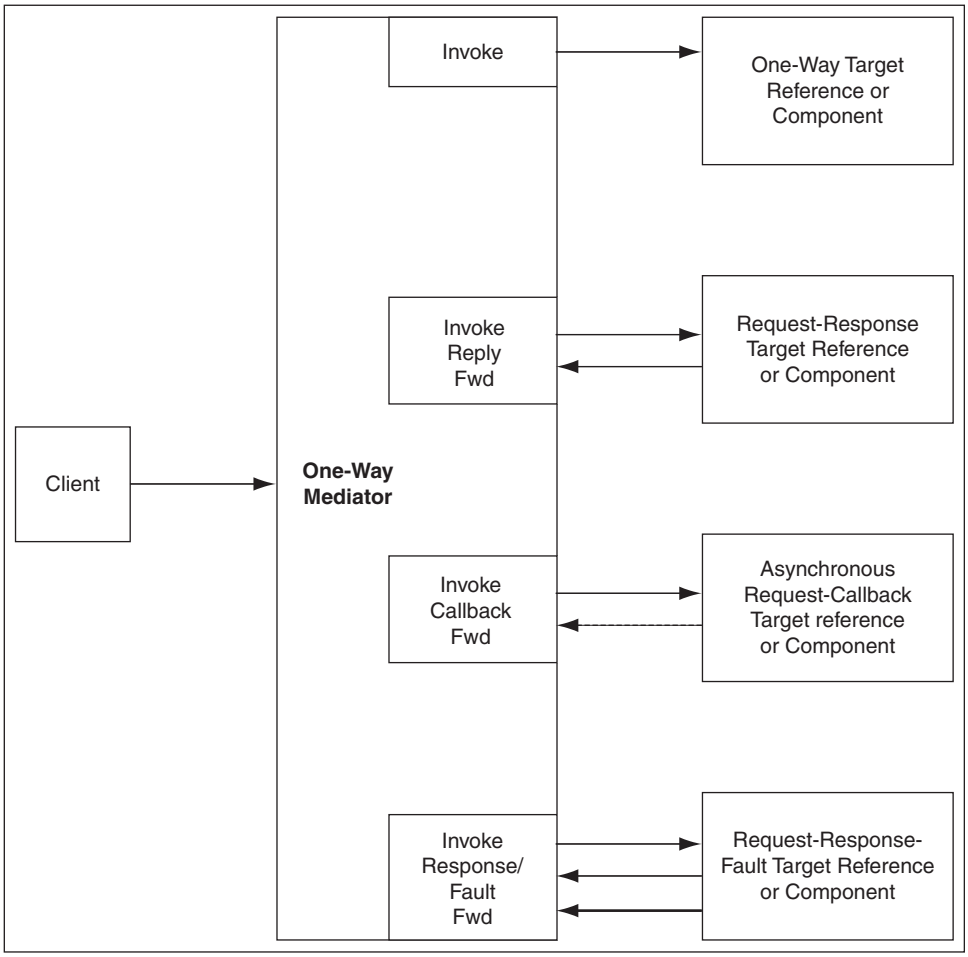
In a one-way interaction, the Oracle Mediator is invoked, but it does not send a response back to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 23–1:

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 one.way.returns.fault Is...	Then...

Set to `true`:
`. . .`
`<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>`
`. . .`
	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 11*g* Release 1.)

Set to `false`:
`. . .`
`<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/LocalSandbox/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>`
`. . .`
	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 11*g* 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

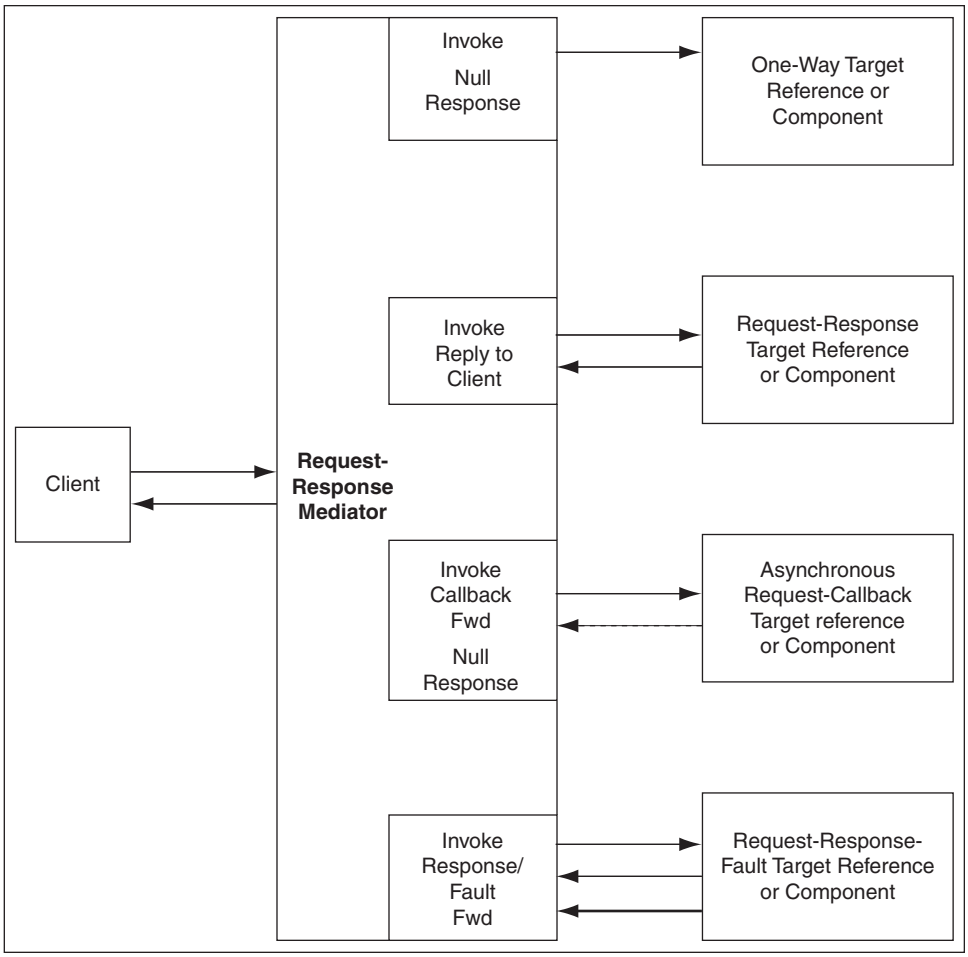
In a request-reply interaction, the Oracle Mediator is invoked and sends a reply to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 23–3:

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

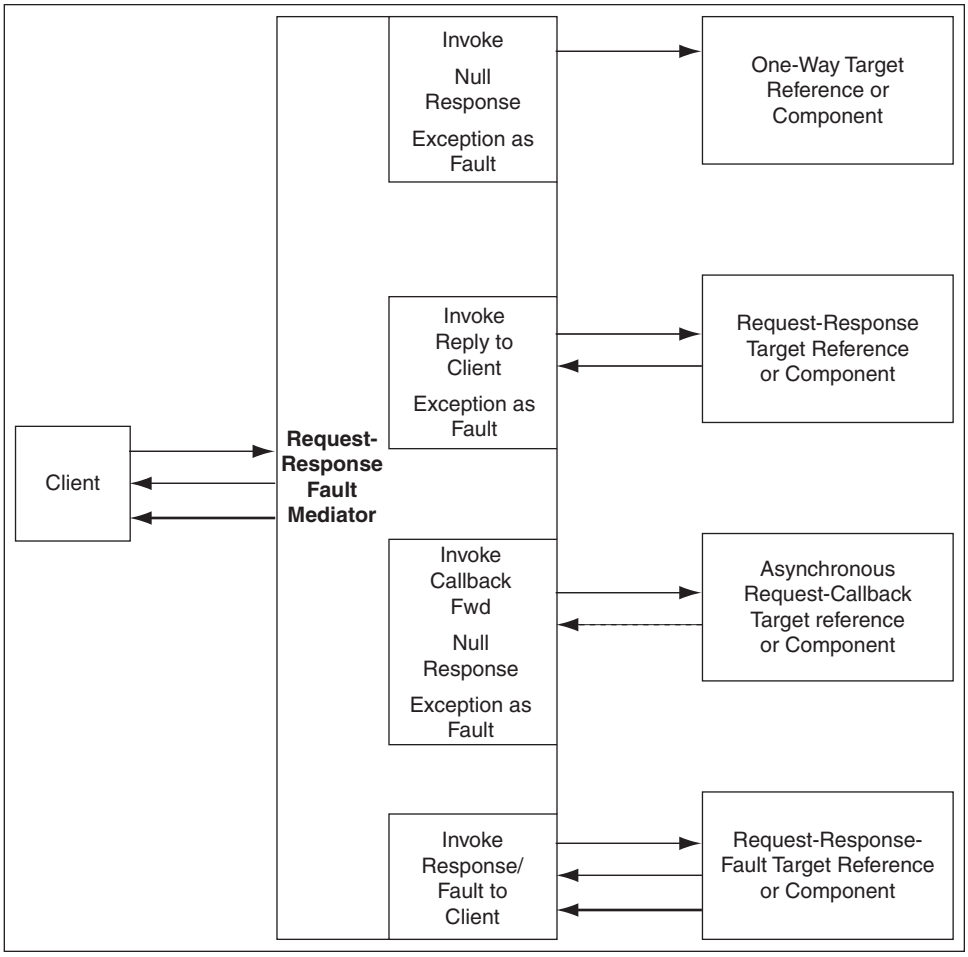
Routing Rule Target Type	Response
Request	There should be at least one sequential routing rule with a request-response-fault service. Oracle Mediator returns `null` when there is no response to be sent.
Request Response	The response is sent back to the caller. Any exception in Oracle Mediator message 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 `null` 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

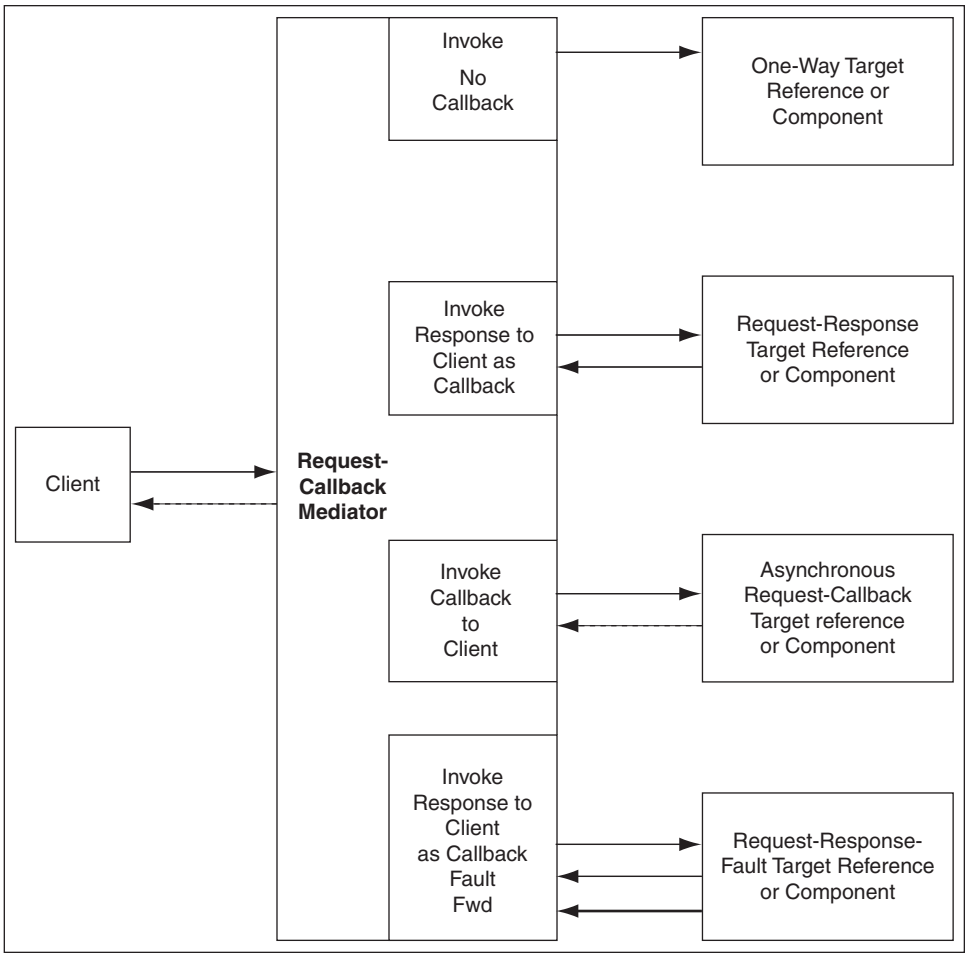
WSDL of the Routing Rule Target	Response
Request	There should be at least one sequential routing rule with a request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.

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

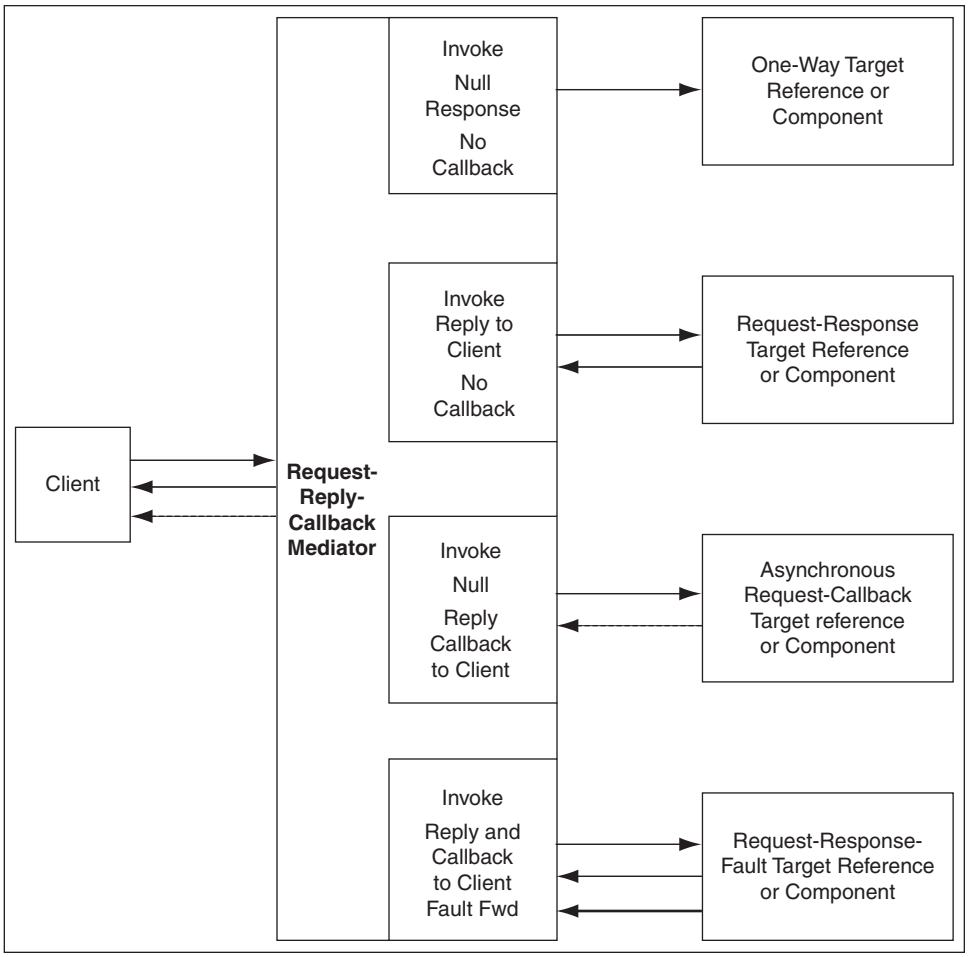
In a request-reply-callback interaction, the Oracle Mediator is invoked and sends a response and an asynchronous reply to the initial caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 23–6:

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 `null` 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

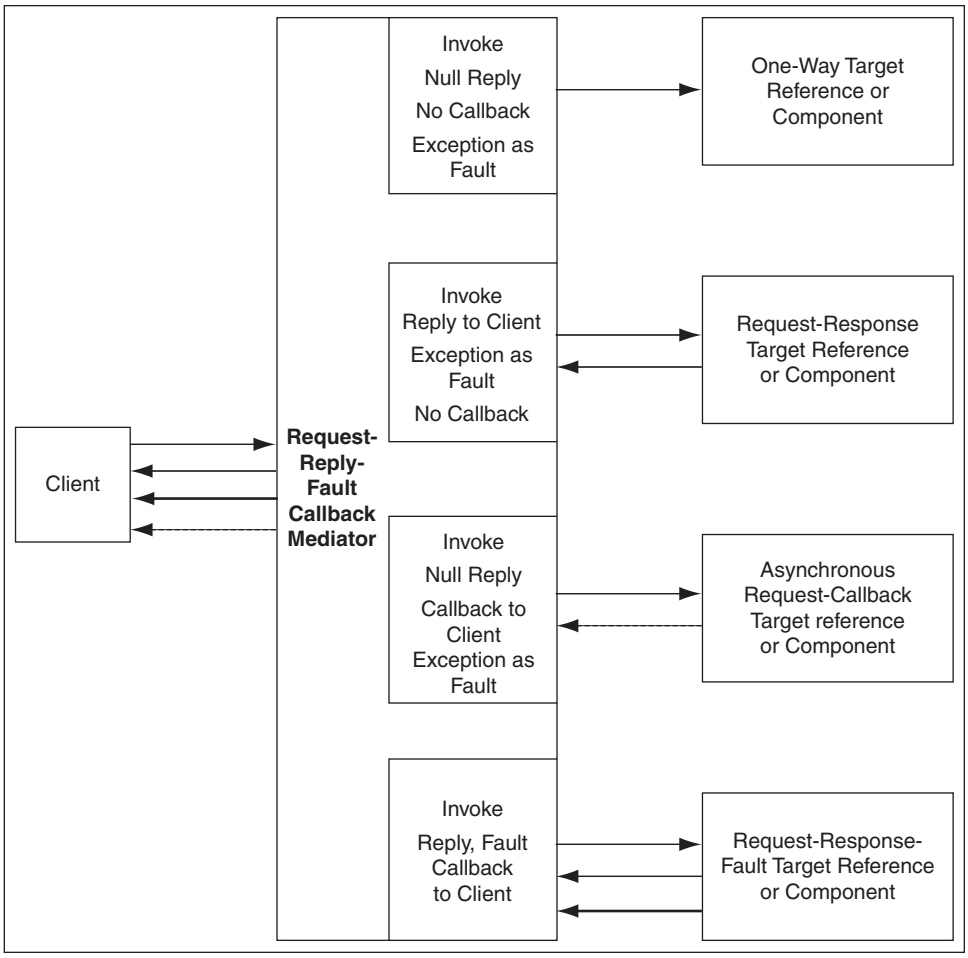
WSDL of the Routing Rule Target	Response
Request	There should be at least one sequential routing rule with a request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.
Request Response	There should be at least one sequential routing rule with a request-callback 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:

24

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:

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.

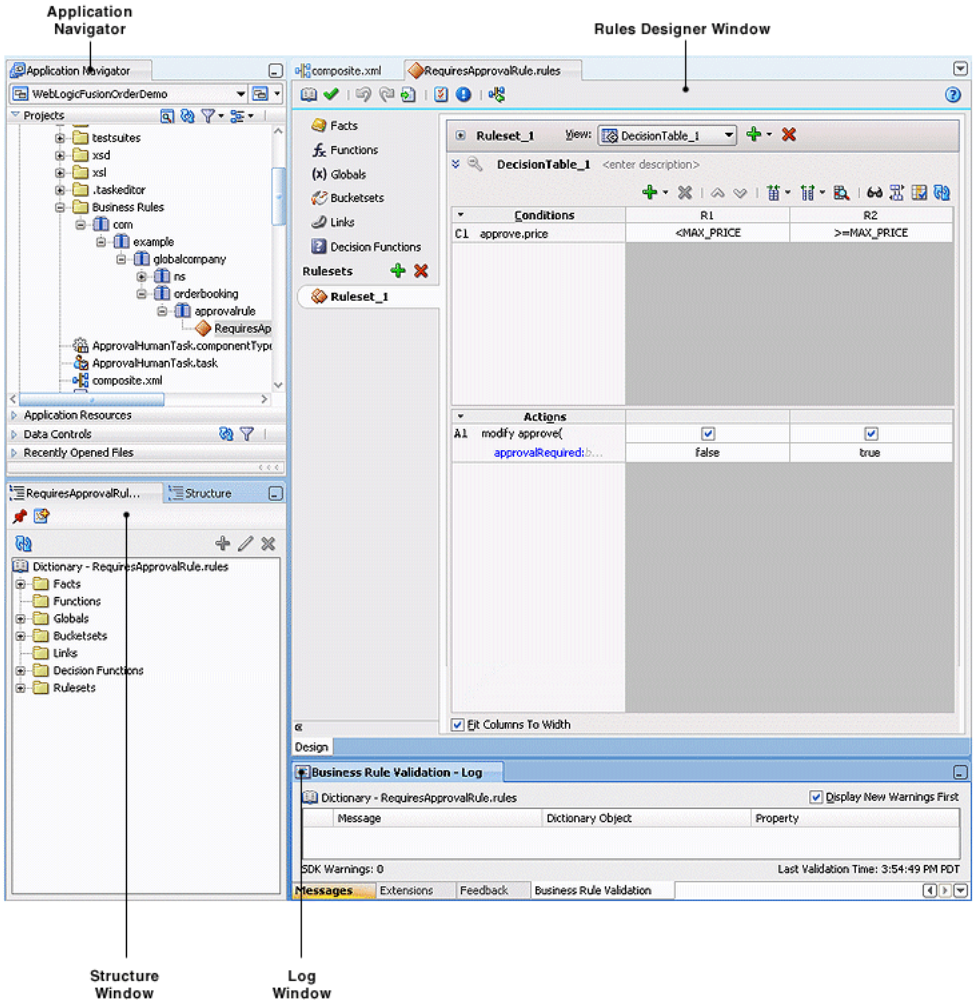
For more information about creating business rules in the Human Task editor of a human task component, see Section 27.3.7.2, "Specifying Advanced Task Routing Using Business Rules."

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*.

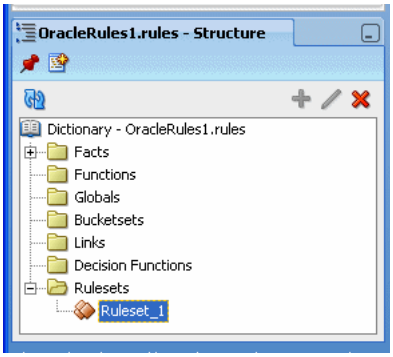
24.2.3 Structure Window

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



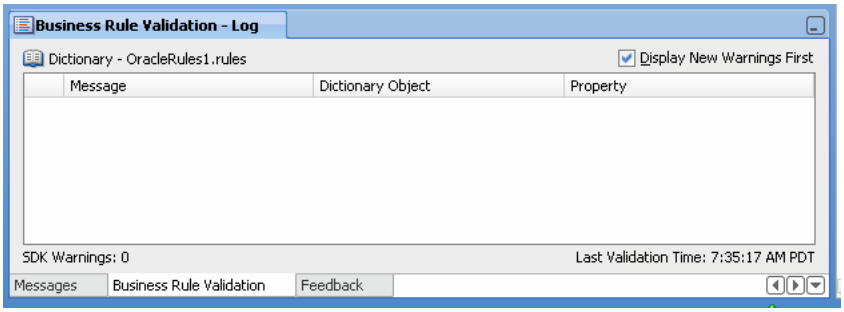
24.2.4 Business Rule Validation Log Window

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 *JDev_Oracle_Home*\JDev\bin\jdev.exe or create a shortcut	$ORACLE_HOME/jdev/bin/jdev

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).

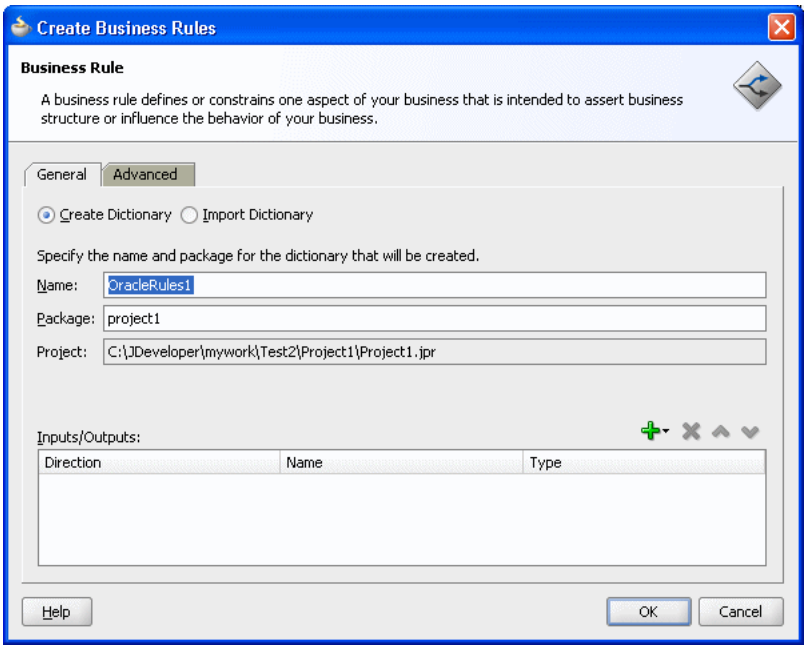
   f. Click **Next**.

   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.

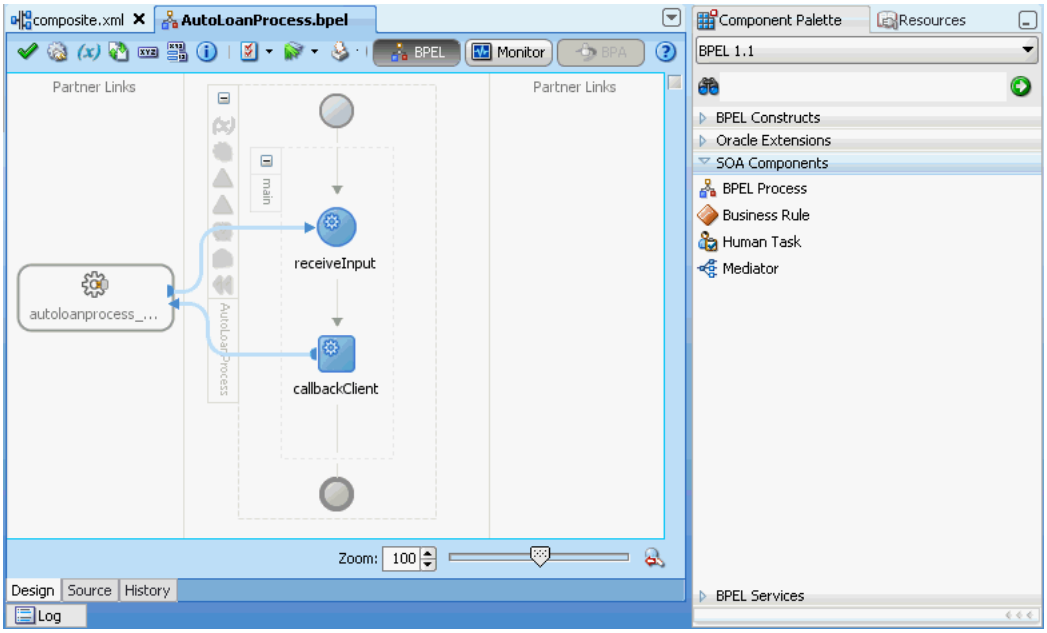
A business rule component enables you to execute business rules and make business decisions based on the rules. To create a business rule component, also called a Decision component, you drag-and-drop a **Business Rule** from the component palette into the BPEL process.

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."

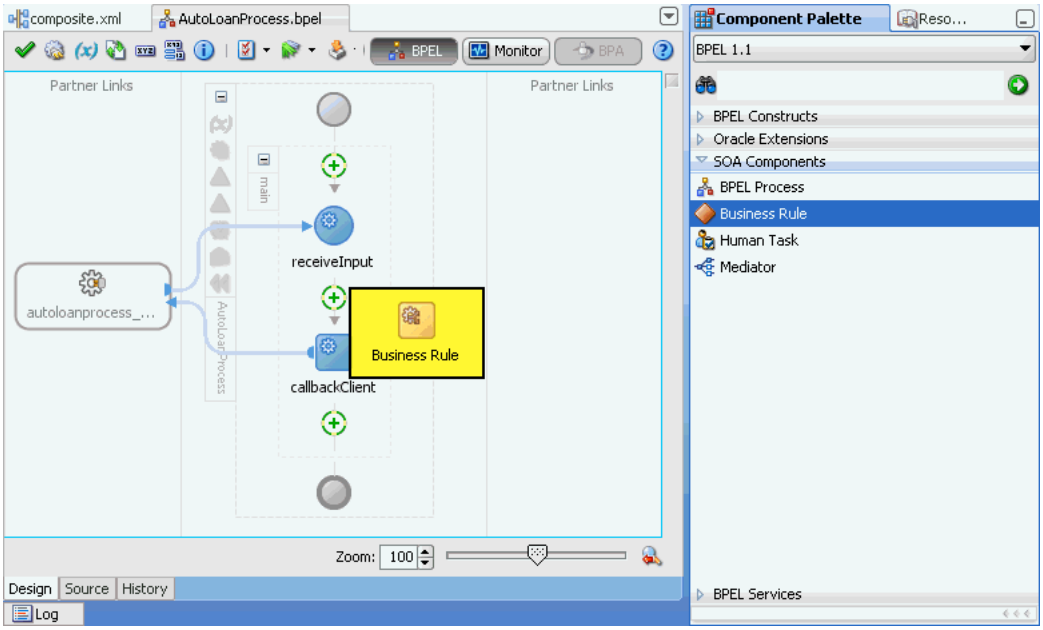
2. Expand the BPEL process by double-clicking the process item. For example, expand the BPEL process to view `receiveInput` and `callbackClient` as shown in Figure 24–5.

Figure 24–5 Adding A Business Rule to a BPEL Process



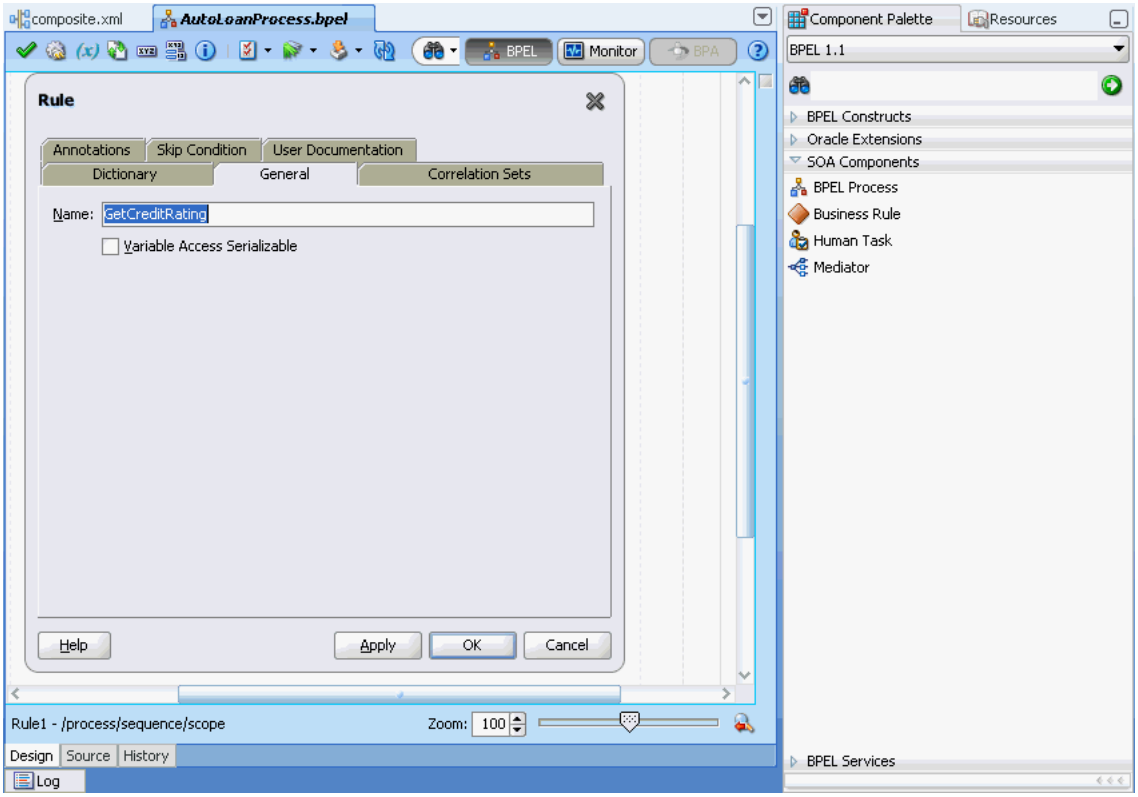
3. Select **Business Rule** from the SOA Components section of the Component Palette and drag-and-drop a **Business Rule** into the position where the business rules are needed. For example, drag-and-drop a **Business Rule** between `receiveInput` and `callbackClient`, as shown in Figure 24–6.

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 to create. 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



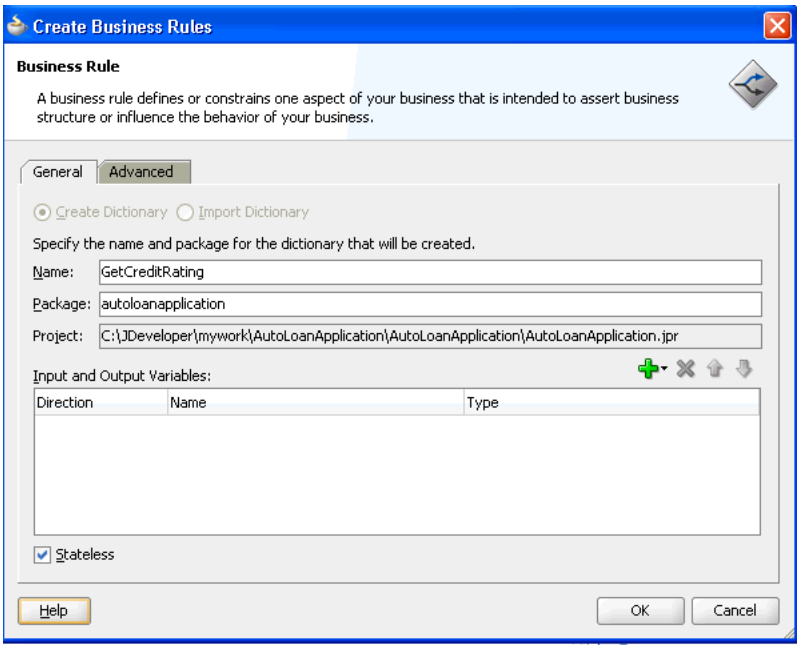
5. In the Business Rule area for the **Business Rule Dictionary**, click the **Create Dictionary** icon to display the Create Business Rules dialog.

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:

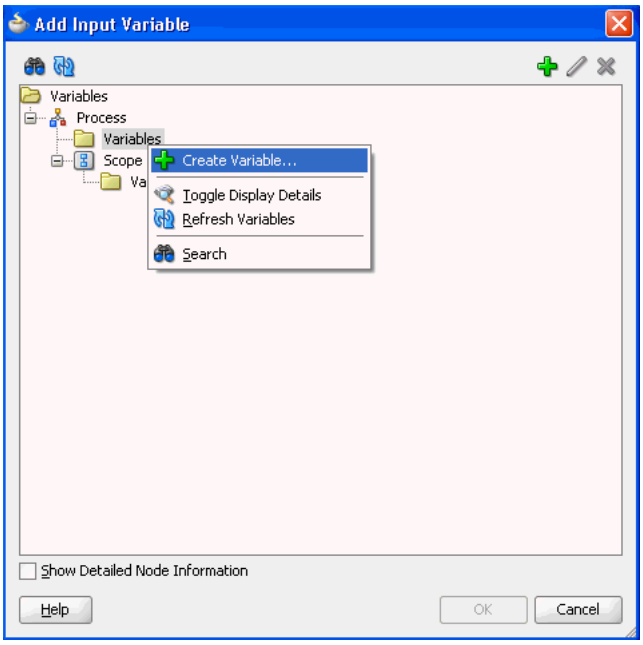
1. In the Create Business Rules dialog, from the menu next to the **Add** icon select **Add Input Variable...** to create the input variable.

   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.

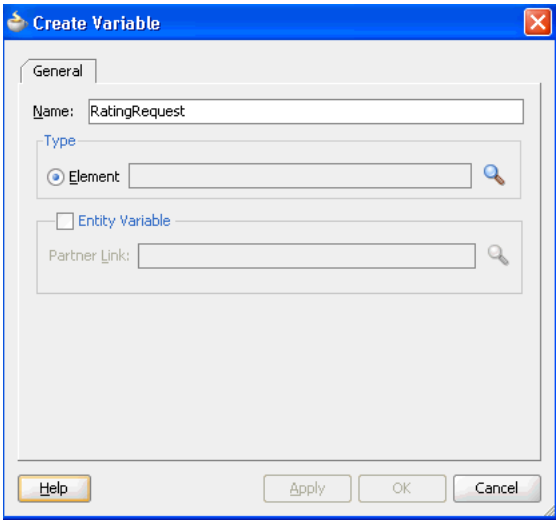
Figure 24–9 Add Input Variable



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**.

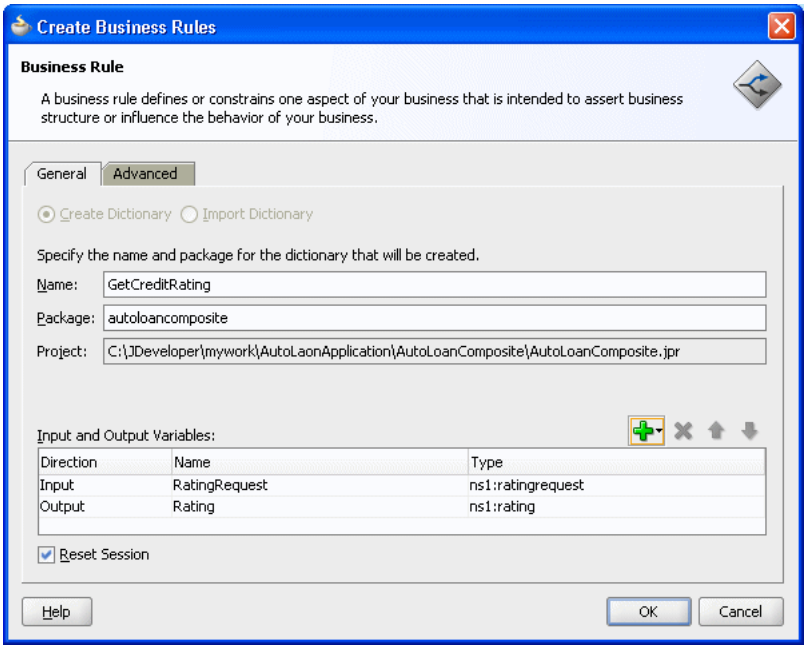
9. In the Add Input Variable dialog, click **OK**.

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



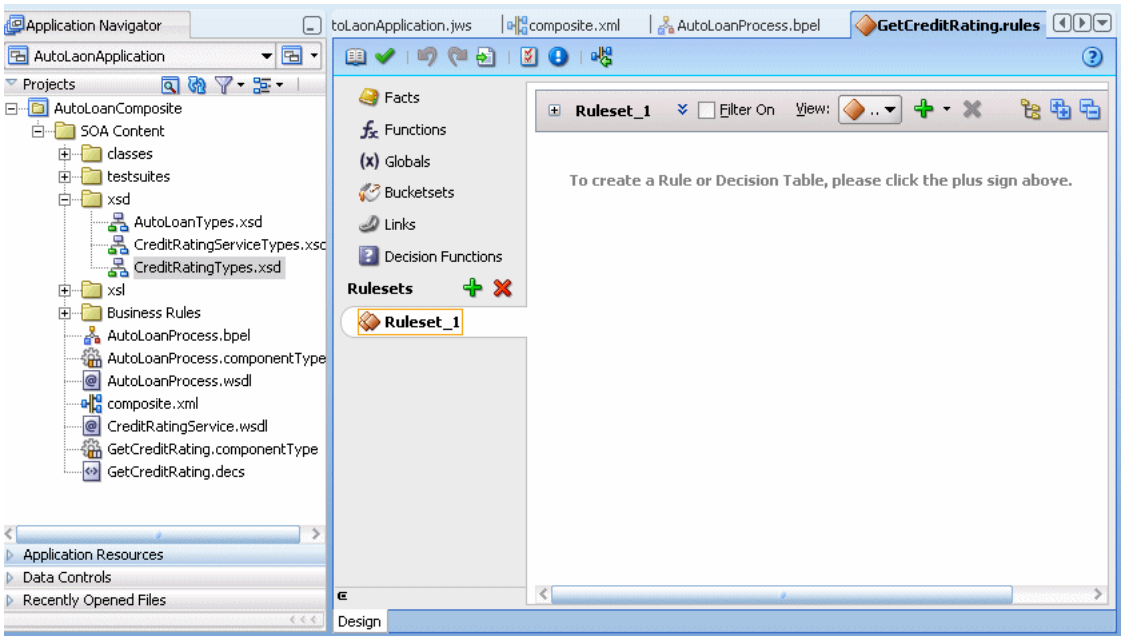
Set options and create decision service and business rules dictionary:

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

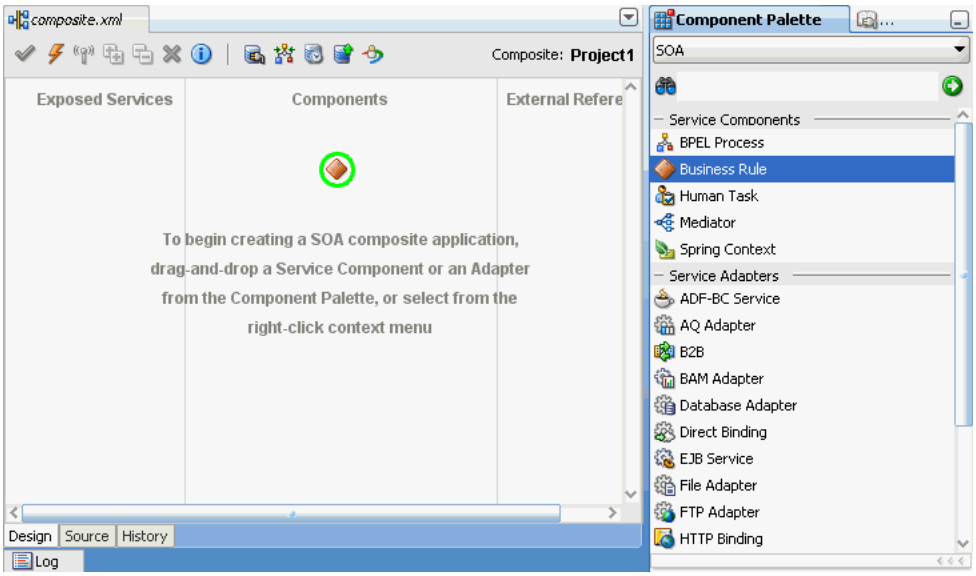
24.5.1 How to Add Business Rules to a SOA Composite Application

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:

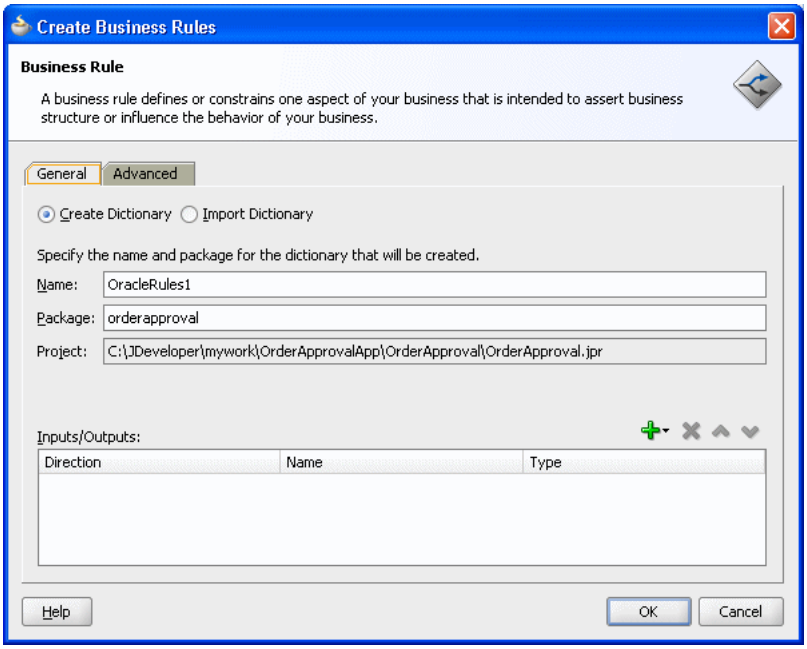
1. Create a SOA application and project. For more information, see Section 2.1.1, "How to Create a SOA Application and Project". For a SOA composite using business rules, pick the required technologies for your application. For example, you may need the following for a SOA application with business rules: 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

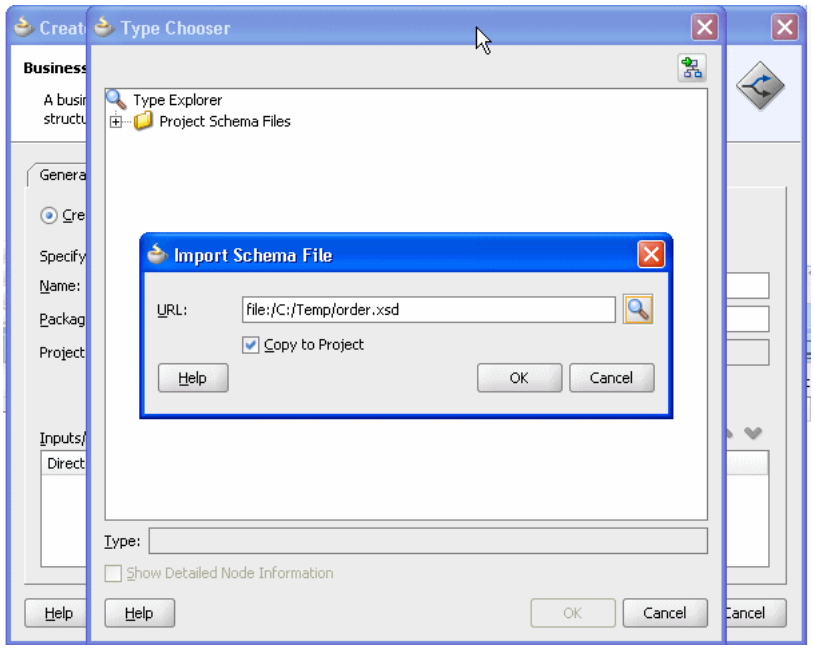


Add inputs for business rules:

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

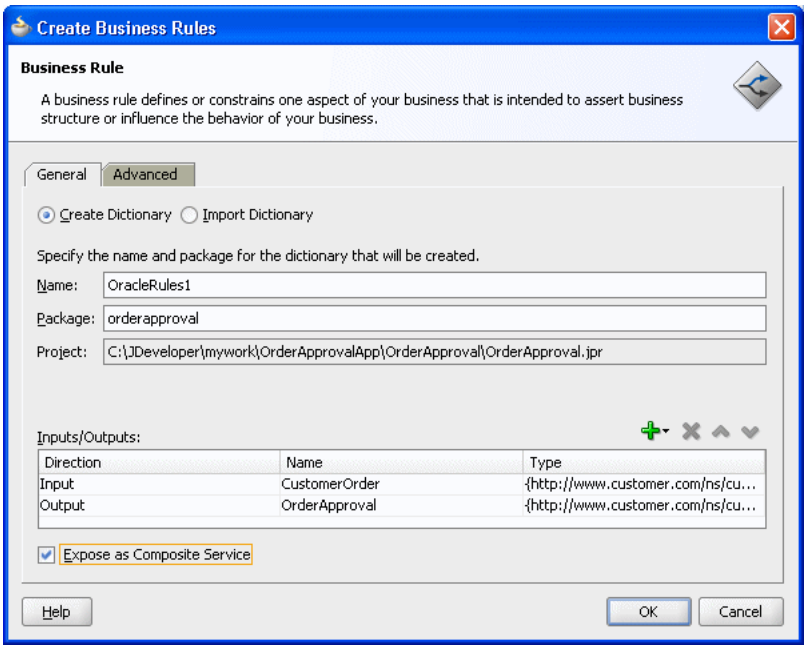


7. In the Import Schema File dialog, click **OK**.
8. In the Localize Files dialog, click **OK**.
9. Use the Type Chooser dialog navigator to locate and select the input from the schema and click **OK**. For example, select the `CustomerOrder` element as the input.

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.

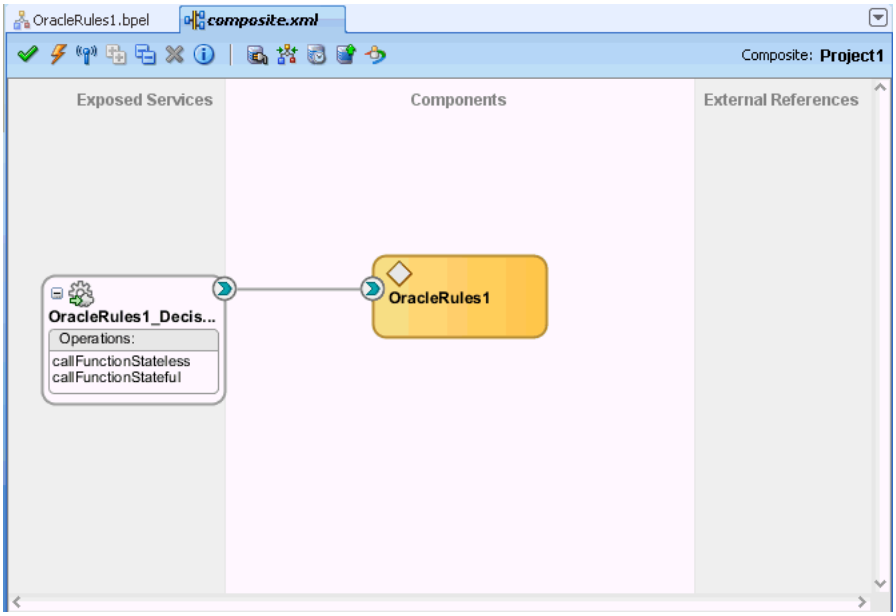
Figure 24–16 Create Business Rules Dialog with Input and Output



Set options and create decision service and business rules dictionary:

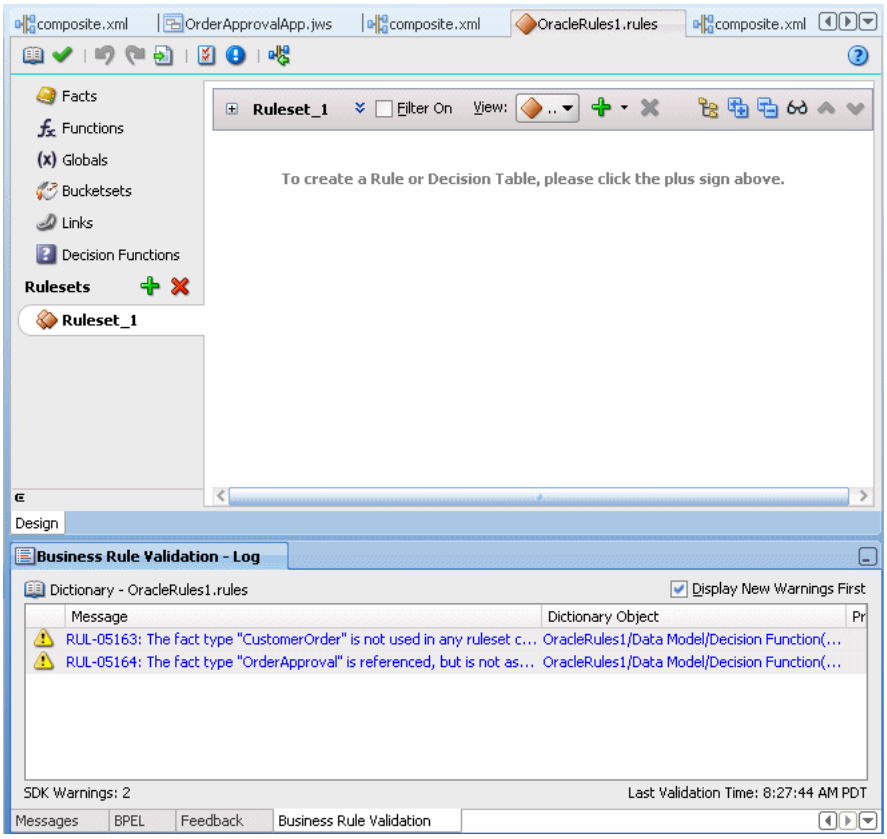
1. In the Create Business Rules dialog, select **Expose as Composite Service**.
2. If you do not want to use the default service name, then select the **Advanced** tab and in the **Service Name** field enter the service name.
3. In the Create Business Rules dialog, click **OK**. This creates the Business 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*.

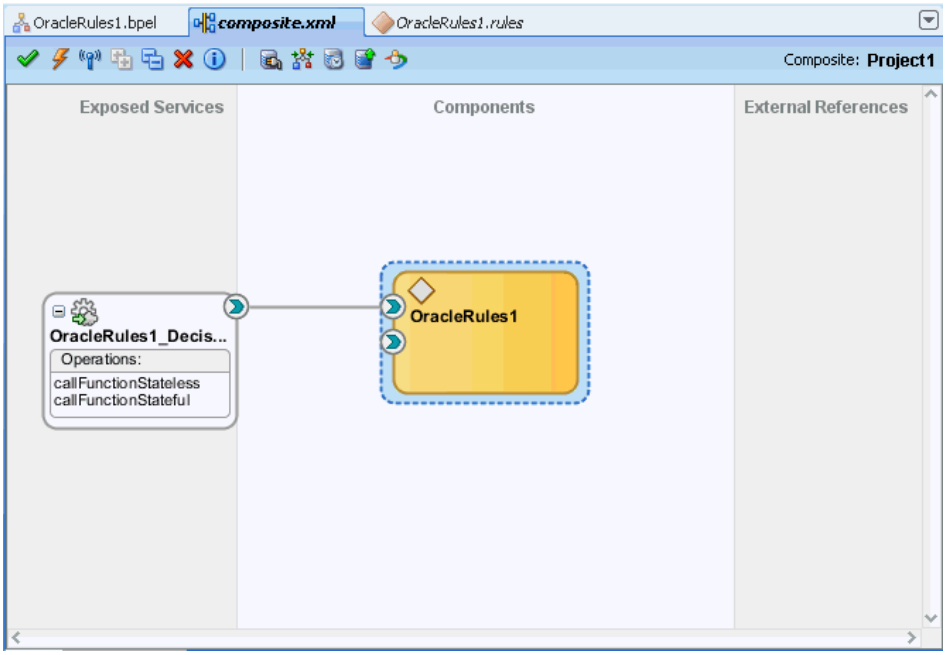
24.5.2 How to Select and Modify a Decision Function in a Business Rule Component

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:

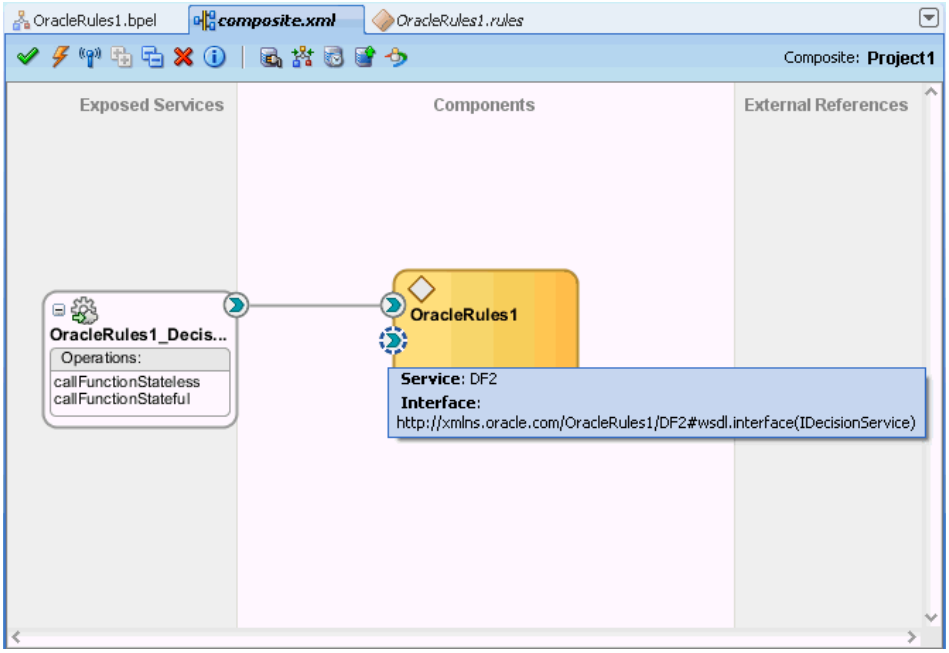
1. Add a decision function to the Oracle Business Rules dictionary. For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.
2. Add a Business Rule component to the composite application. For more information, see Section 24.5.1, "How to Add Business Rules to a SOA 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



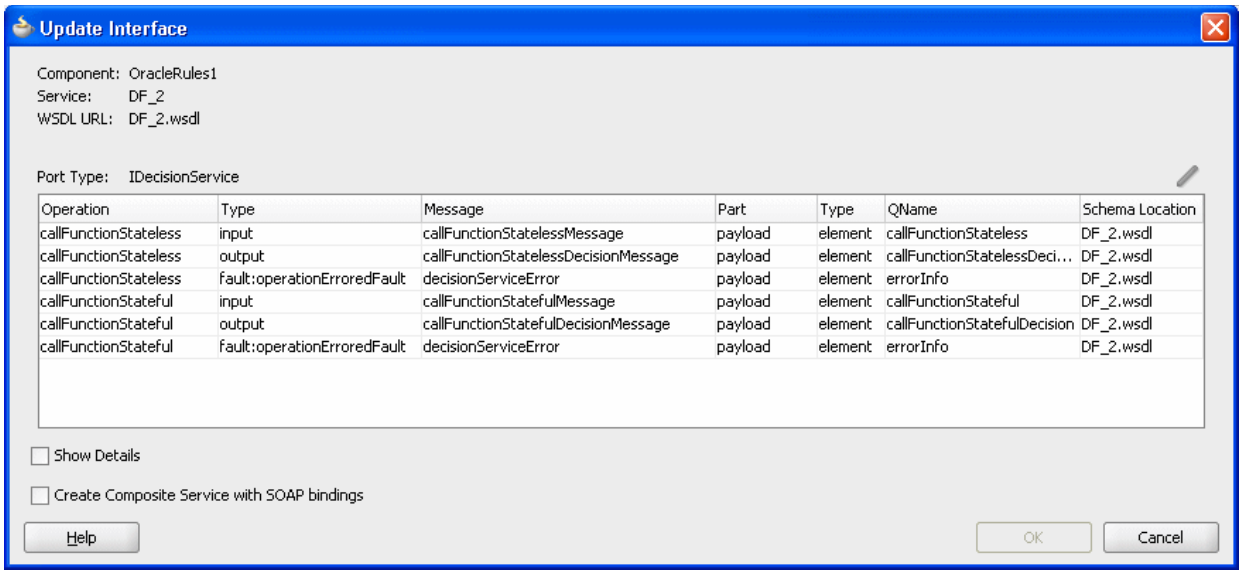
4. Select the decision function port of interest. For example, select the port for DF_2 as shown in Figure 24–20.

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



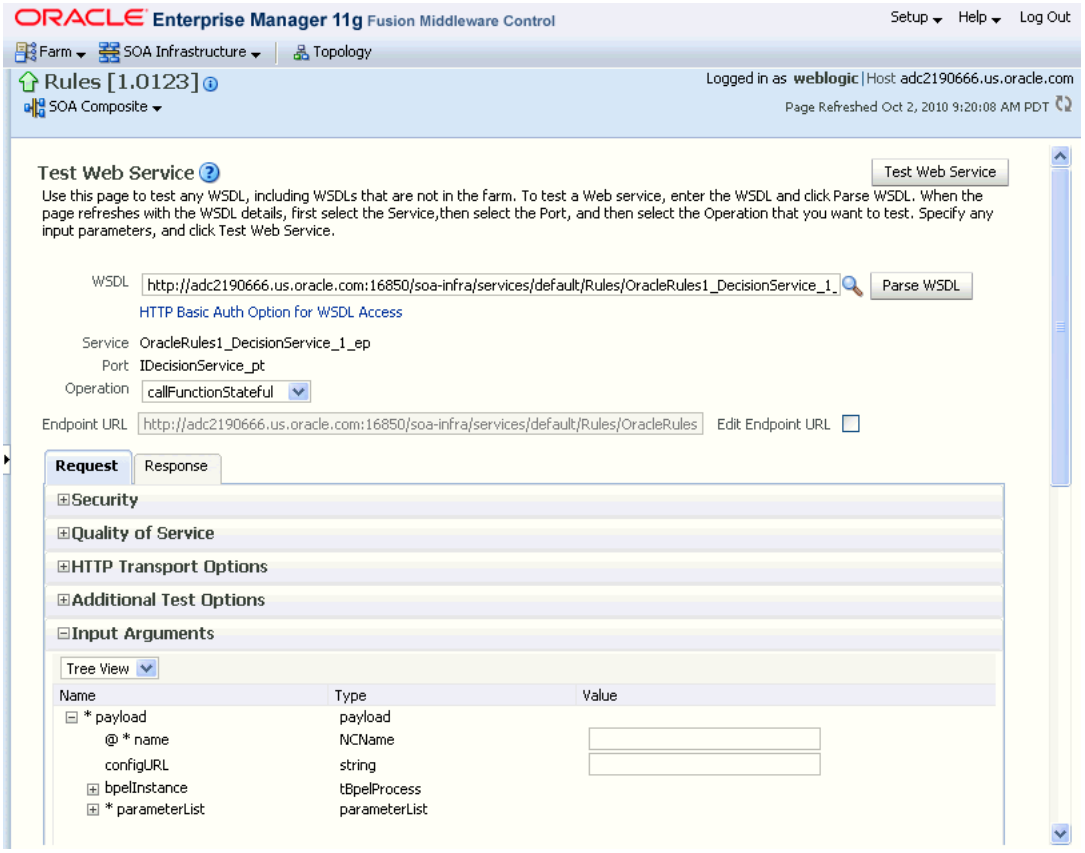
24.6 Running Business Rules in a Composite Application

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



'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`

25

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:

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.

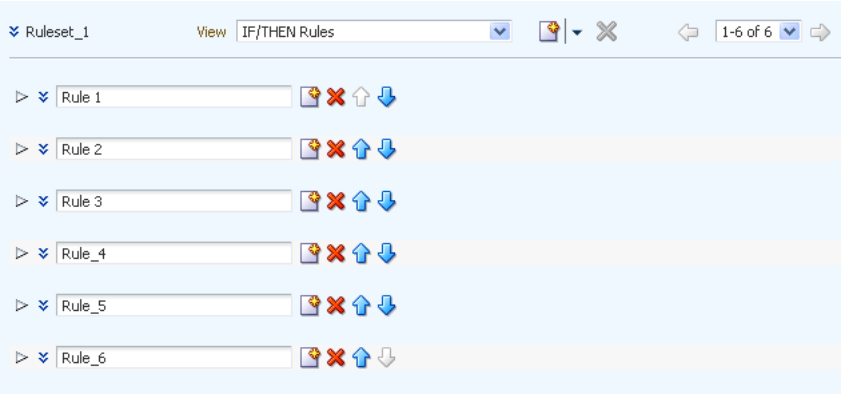
Note: You should not confuse Rules Editor with Rules Dictionary Editor. Rules Editor is used to edit rules inside a specified ruleset. In fact, Rules Editor is embedded within Rules Dictionary Editor. For more information about Rules Dictionary Editor, see Section 25.3, "Using the Oracle Business Rules 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:

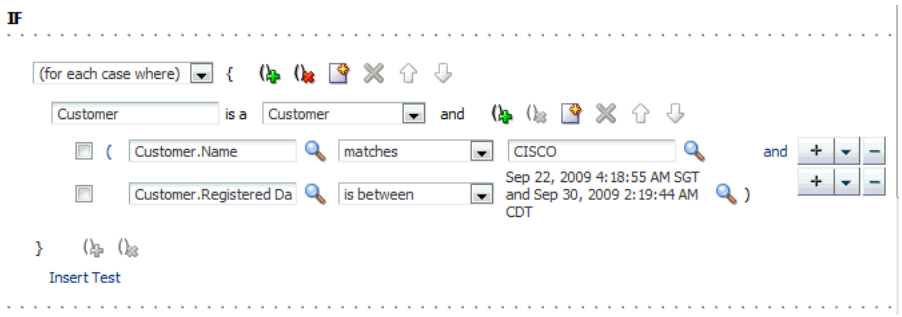
- Creates, updates, and deletes:
  - Rules in a ruleset, as shown in Figure 25–1:

Figure 25–1 Rules in a Ruleset



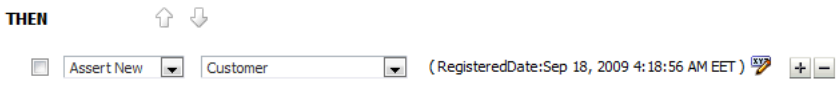
  - Simple tests or conditions in a rule, as shown in Figure 25–2:

Figure 25–2 Simple Tests or Conditions in a Rule



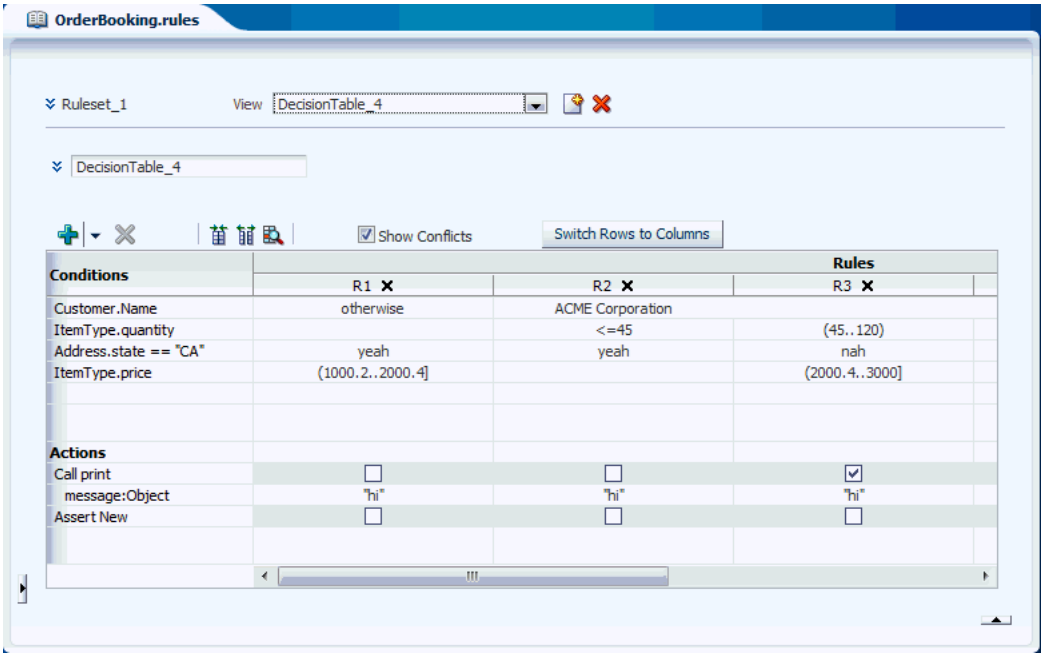
– Actions in a rule, as shown in Figure 25–3.

Figure 25–3 Actions in a Rule



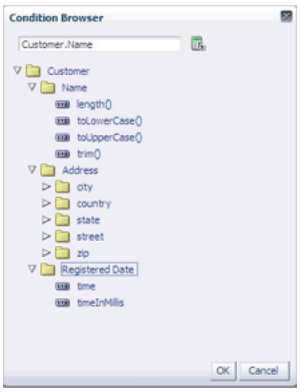
– Decision tables, as shown in Figure 25–4.

Figure 25–4 Decision Tables



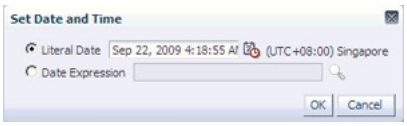
- Sets effective dates and priorities for rulesets and rules.
- Provides support for user-defined operators.
- Provides a Condition Browser pop-up to display the left or right value options, as shown in Figure 25–5.

Figure 25–5 Condition Browser



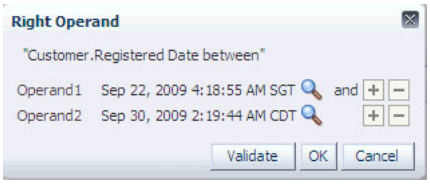
- Provides a Date Browser for selecting date types, as shown in Figure 25–6.

Figure 25–6 Date Browser



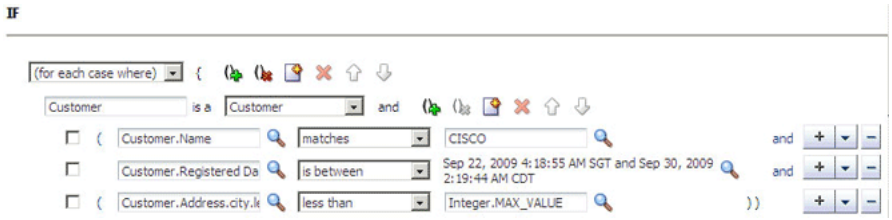
- Provides a Right Operand browser to handle multiple right-hand side expressions, as shown in Figure 25–7.

Figure 25–7 Right Operand Browser



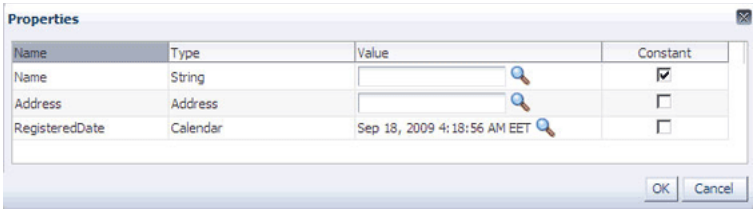
- Provides support for nested rules, as shown in Figure 25–8.

Figure 25–8 Nested Rules Support



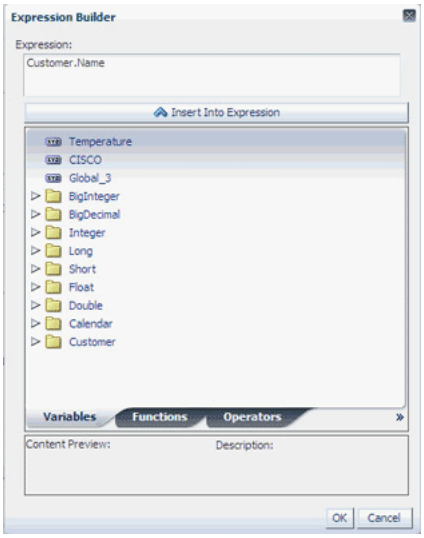
- Provides the Properties browser for editing properties of a rule action, as shown in Figure 25–9.

Figure 25–9 Properties Browser



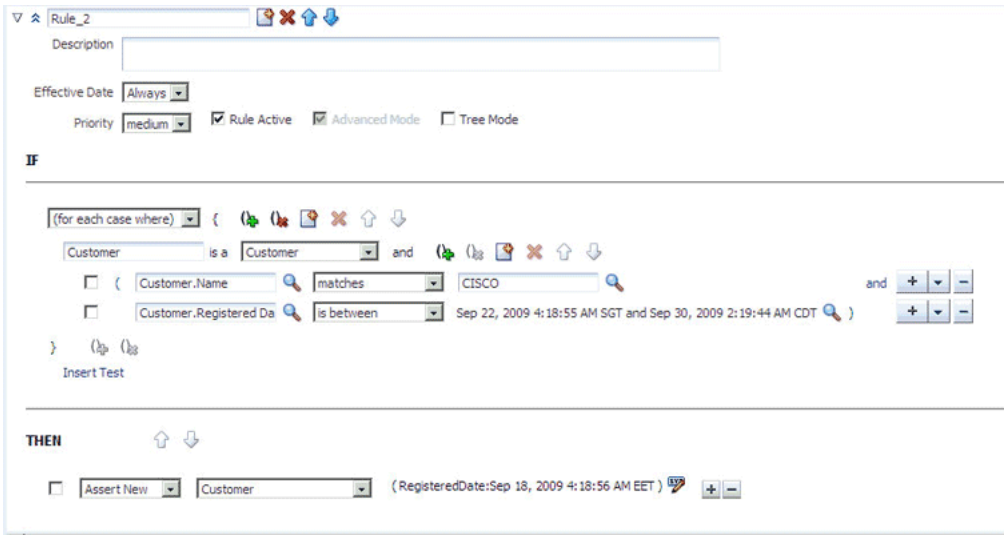
- Provides an Expression Builder window to build custom expressions, as shown in Figure 25–10.

Figure 25–10 Expression Builder Window



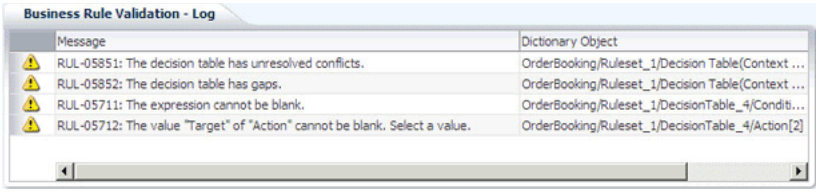
- Provides **Advanced Mode** features for working with patterns and advanced actions, as shown in Figure 25–11.

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.

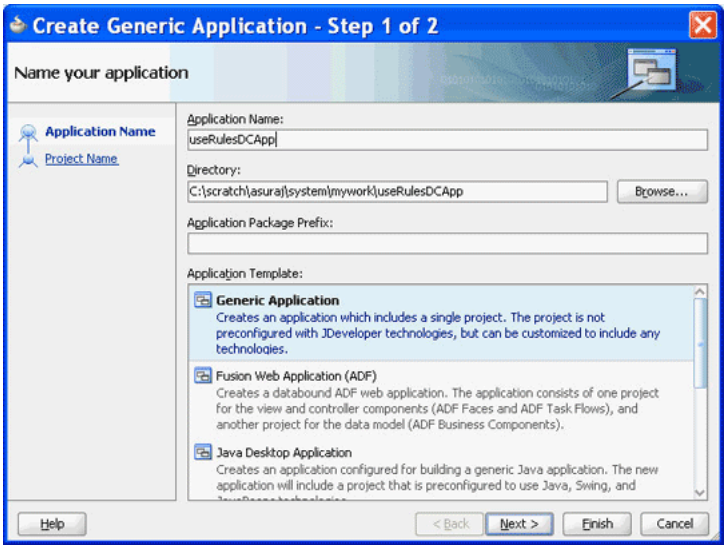
To create a sample application by using the Rules Editor:

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.

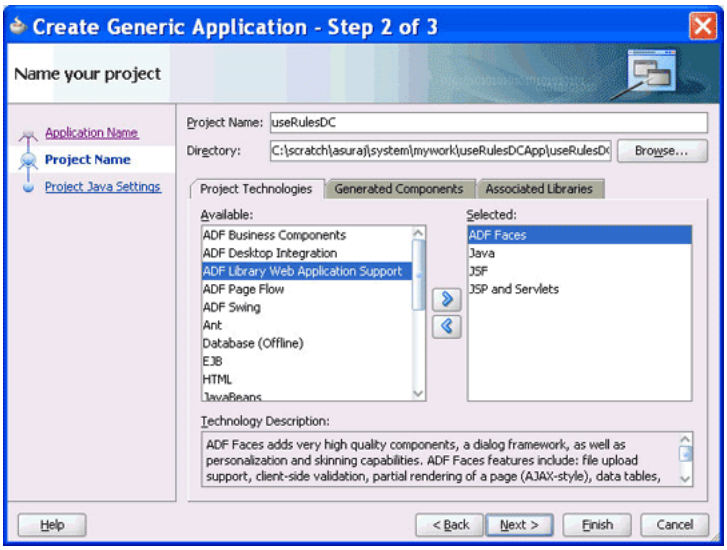
Figure 25–13 Creating a Generic Application



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



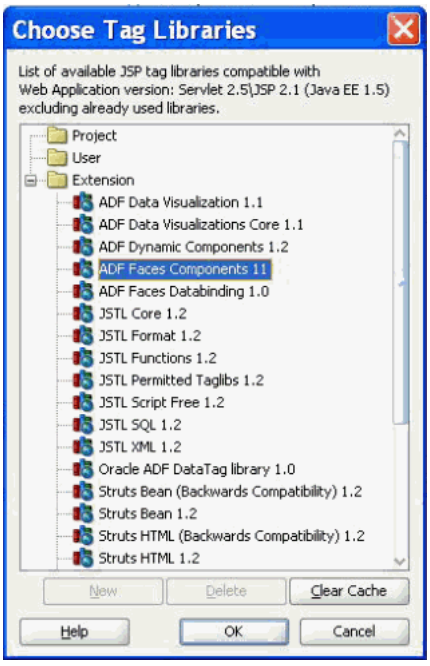
5. Right-click the **useRulesDC** project in the Application Navigator of Oracle JDeveloper, and select **Project Properties** to display the **Project Properties** dialog box.

   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.

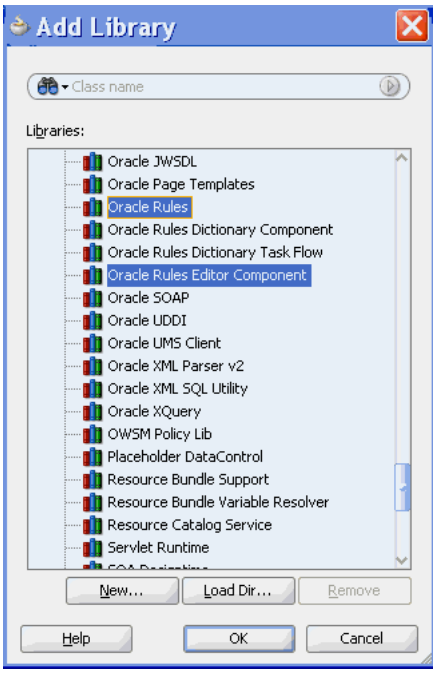
Figure 25–15 Choosing Tab Libraries



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.

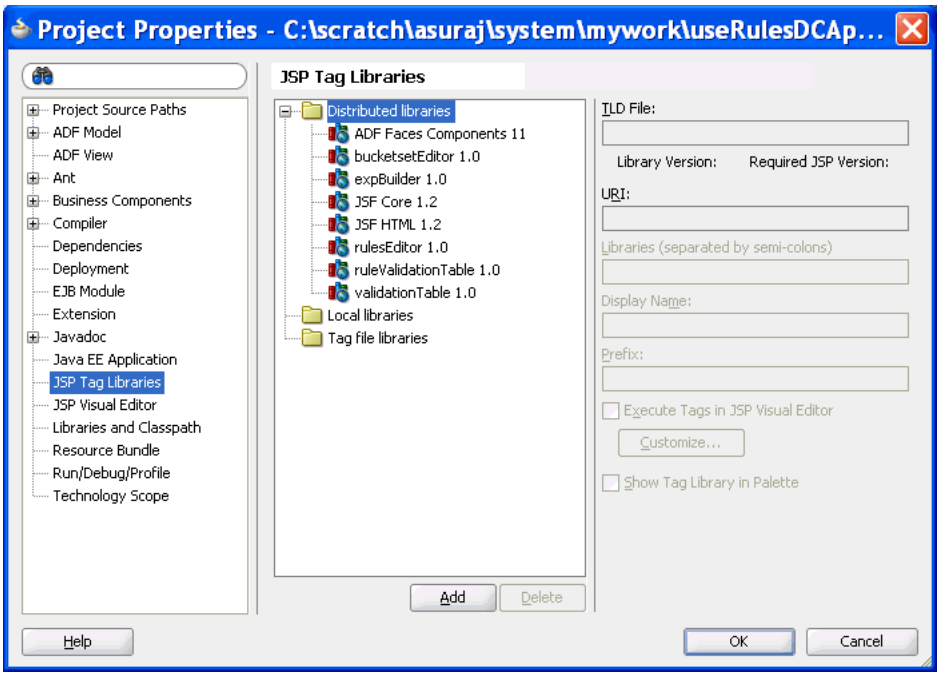
e. Click **OK** to close the Project Properties dialog box.

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



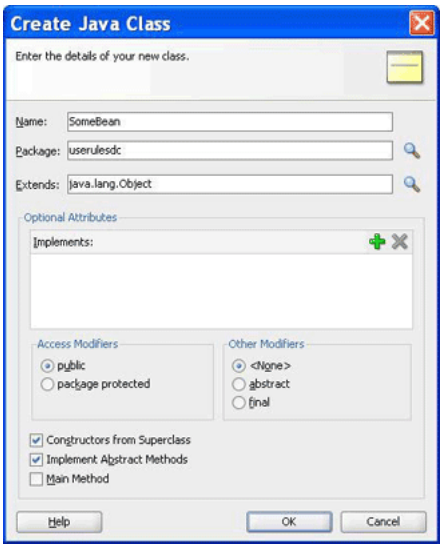
To create the RuleSetModel object:

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();
```

6. 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.

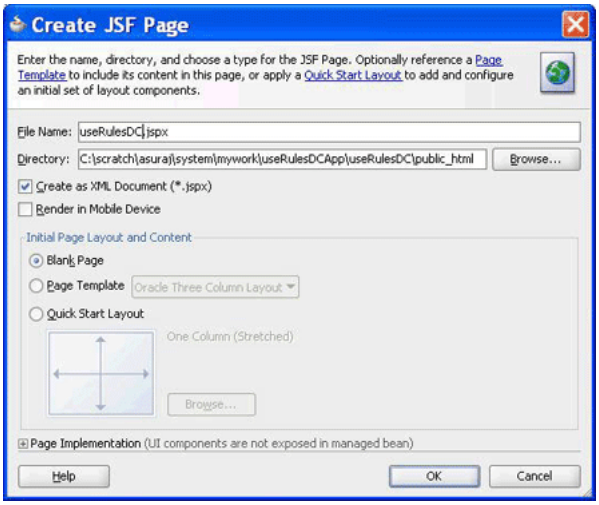
To create the .jspx file for the Rules Editor Component tag:

The next task is to create the .jspx file to include the Rules Editor component tag.

The steps are:

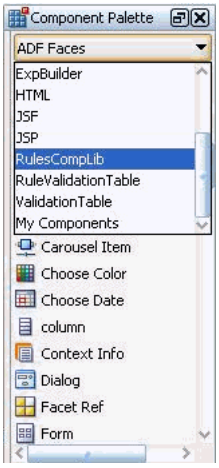
1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** to display the **New Gallery** dialog box.
3. In the **New Gallery** dialog box, select **JSF** under Web Tier from the Categories panel.
4. Select **JSF Page** under Items and click **OK** to display the **Create JSF Page** dialog box.
5. In the **Create JSF Page** dialog box, enter `useRulesDC.jspx` 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:

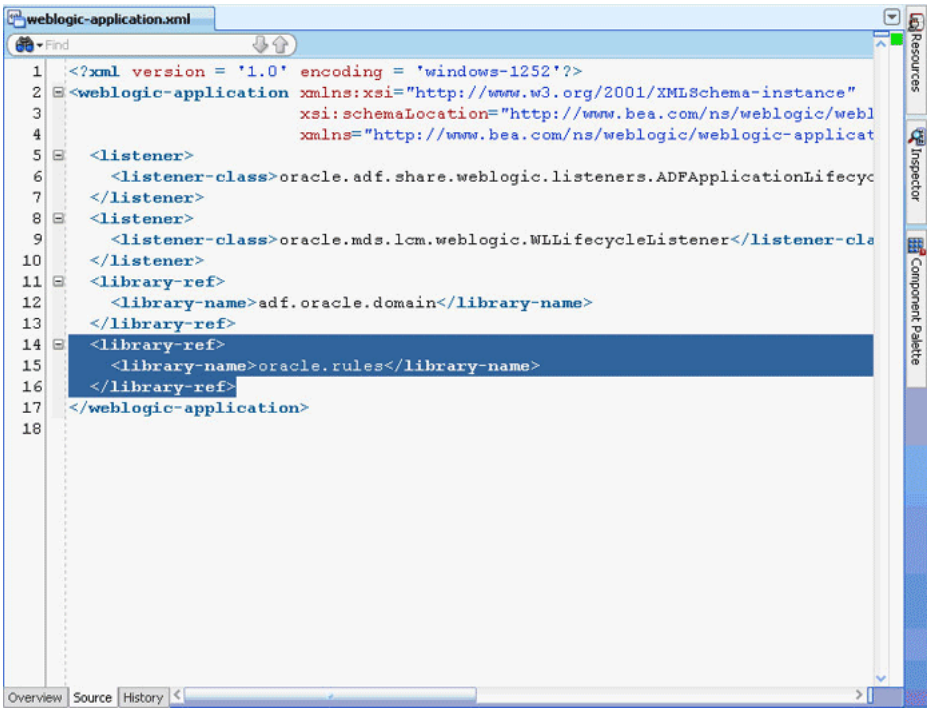
After creating the .jspx file, you must refer to the `oracle.rules` and `oracle.soa.rules_editor_dc.webapp` shared libraries from the weblogic-application.xml file.

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.

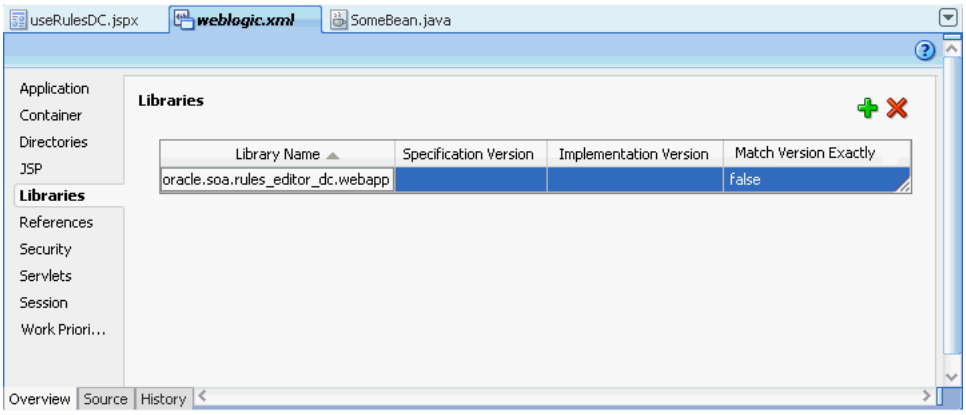
```
<library-ref>
<library-name>oracle.rules</library-name>
</library-ref>
```

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**.

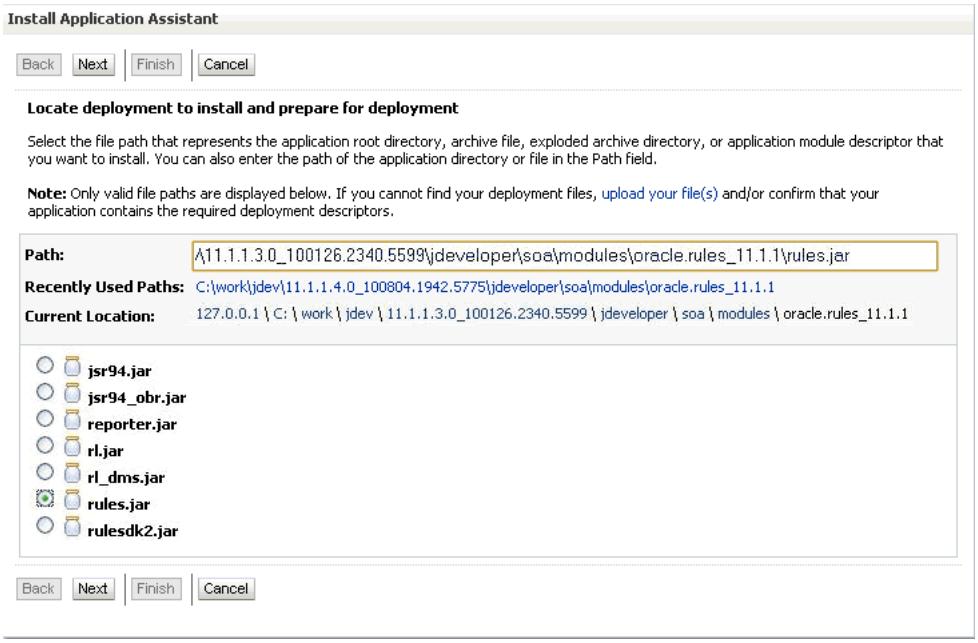
4. Deploy the `oracle.rules` shared library to the embedded Weblogic server:

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



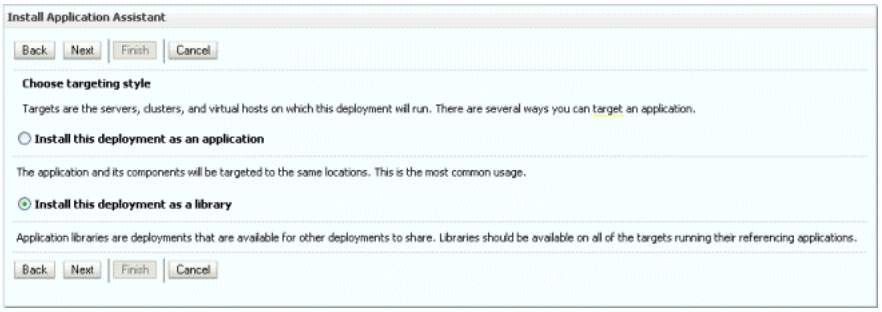
5. Deploy the `oracle.soa.rules_editor_dc.webapp` shared library to the Weblogic server:

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.

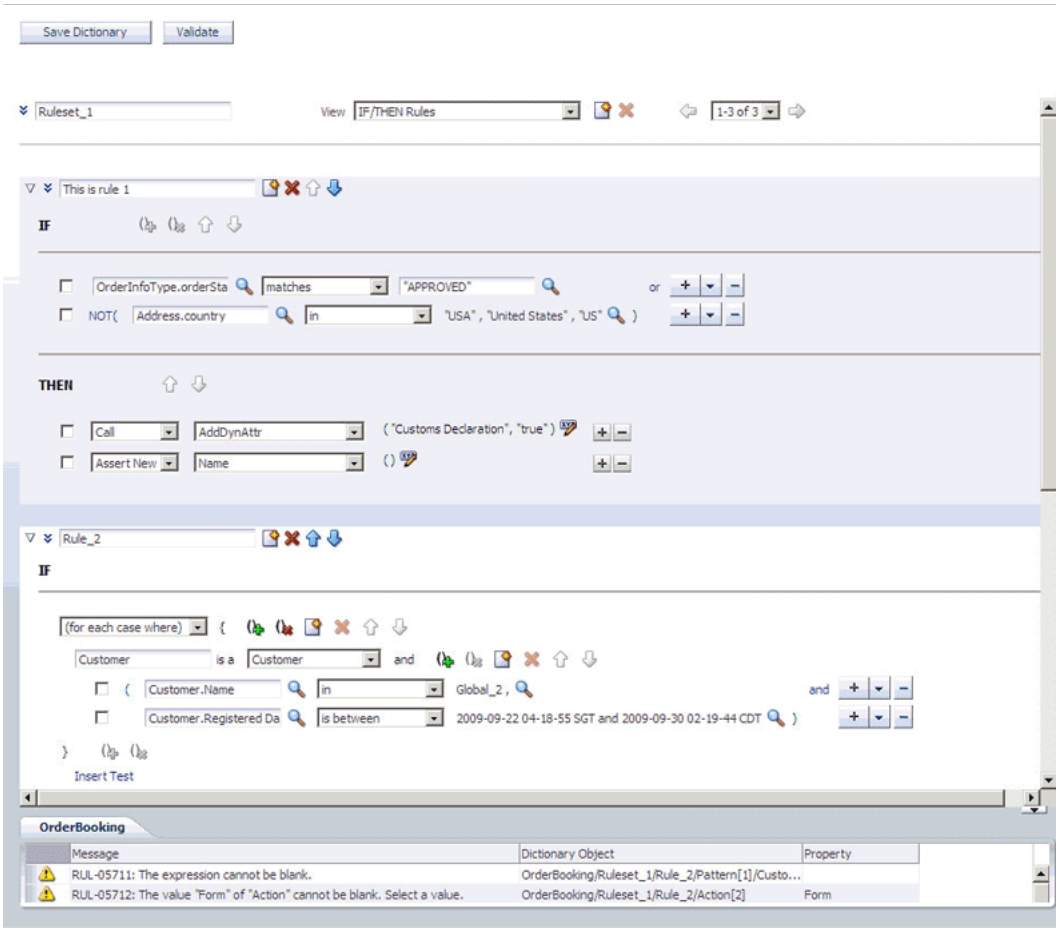
Figure 25–26 oracle.soa.rules_editor_dc.webapp Added to the Deployment List

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

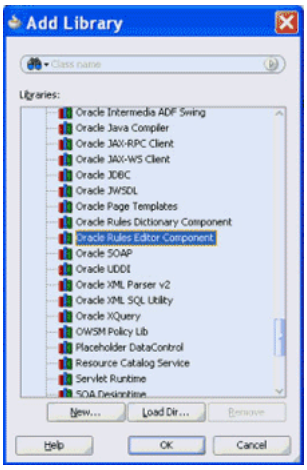


25.2.3 How to Deploy a Rules Editor Application to a Standalone Weblogic Server

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.AddDTPermission`: 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.DeleteDTPermission`: 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.EditDTPermission`: 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.rulesdc.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.rulesdc.model.impl.SimpleTestModel	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.

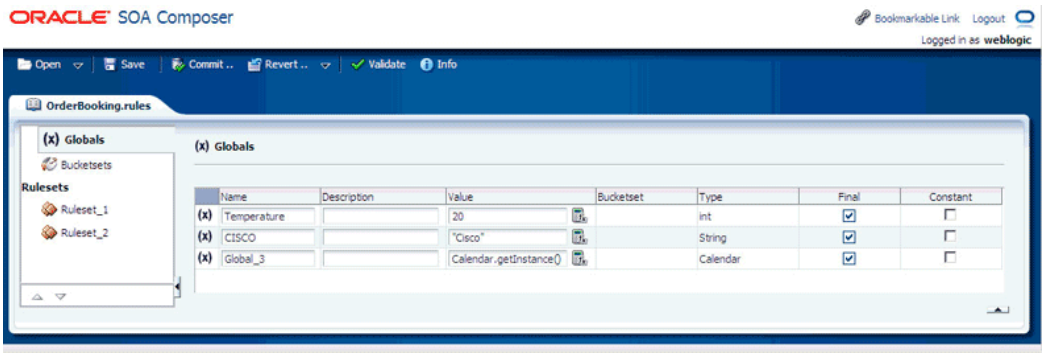
Note: You should not confuse Rules Dictionary Editor with Rules Editor. Rules Editor is used to edit rules inside a specified ruleset. In fact, Rules Editor is embedded within Rules Dictionary Editor. For more information about Rules Editor, see Section 25.2, "Using the Oracle Business Rules Editor Declarative 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.

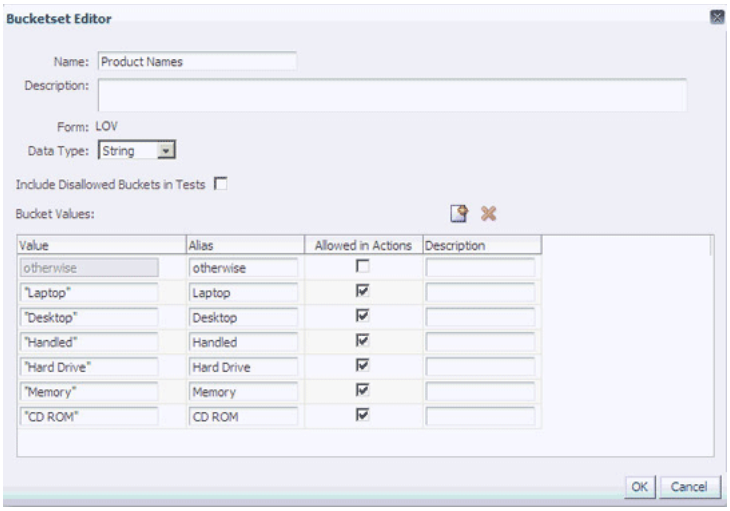
Figure 25–29 Globals Editor



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.

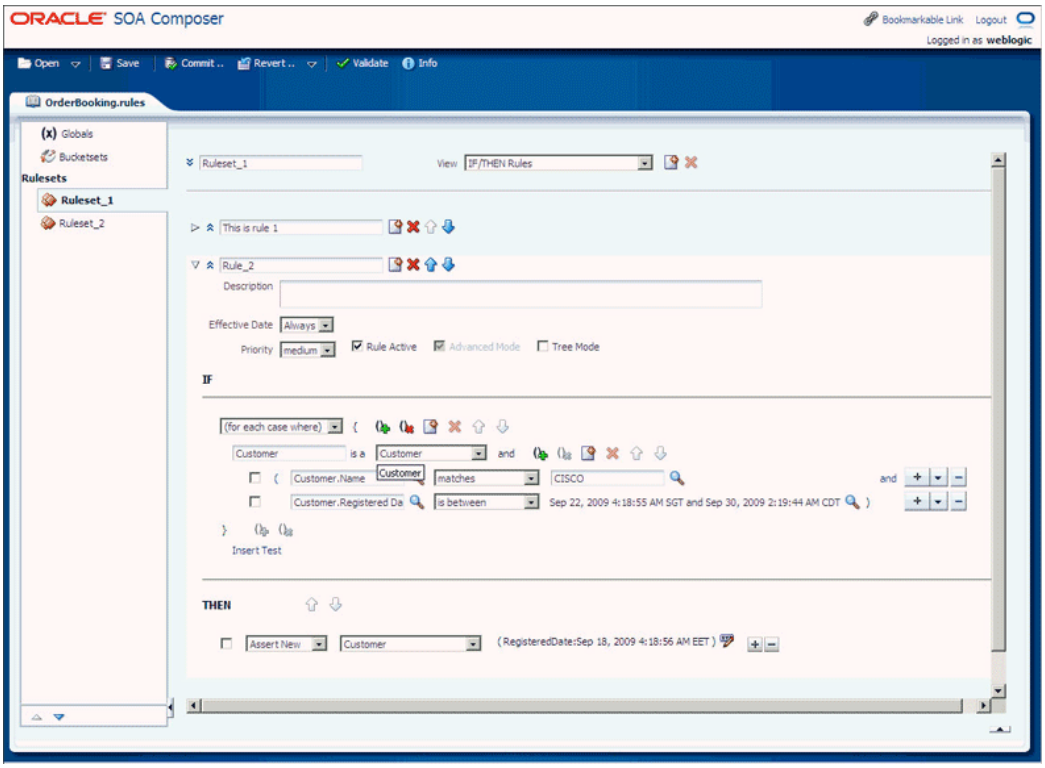
Figure 25–30 Bucketset Editor



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.

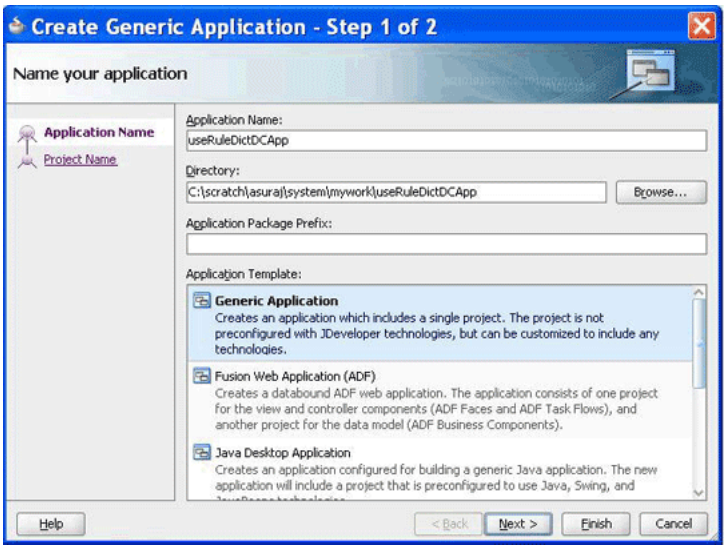
To create a sample application by using the Rules Dictionary Editor:

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.

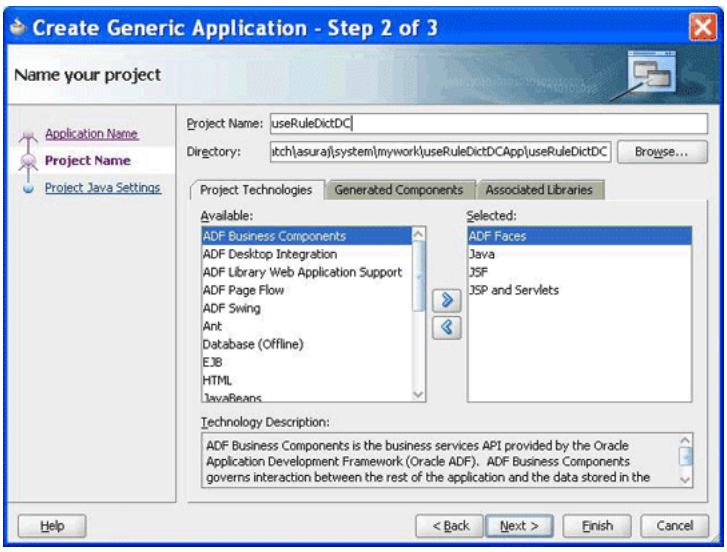
Figure 25–32 Creating a Generic Application



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



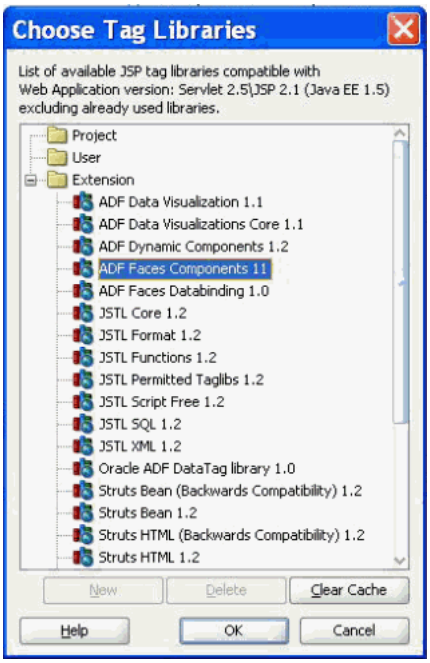
5. Right-click the **useRuleDictDC** project in the Application Navigator of Oracle JDeveloper, and select **Project Properties** to display the **Project Properties** dialog box.

   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.

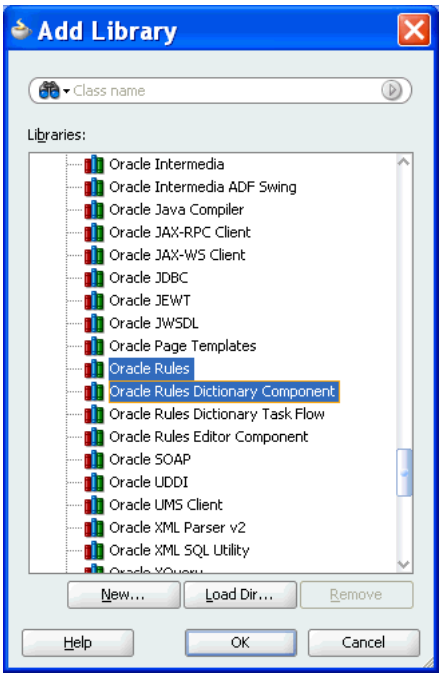
Figure 25–34 Choosing Tab Libraries



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.

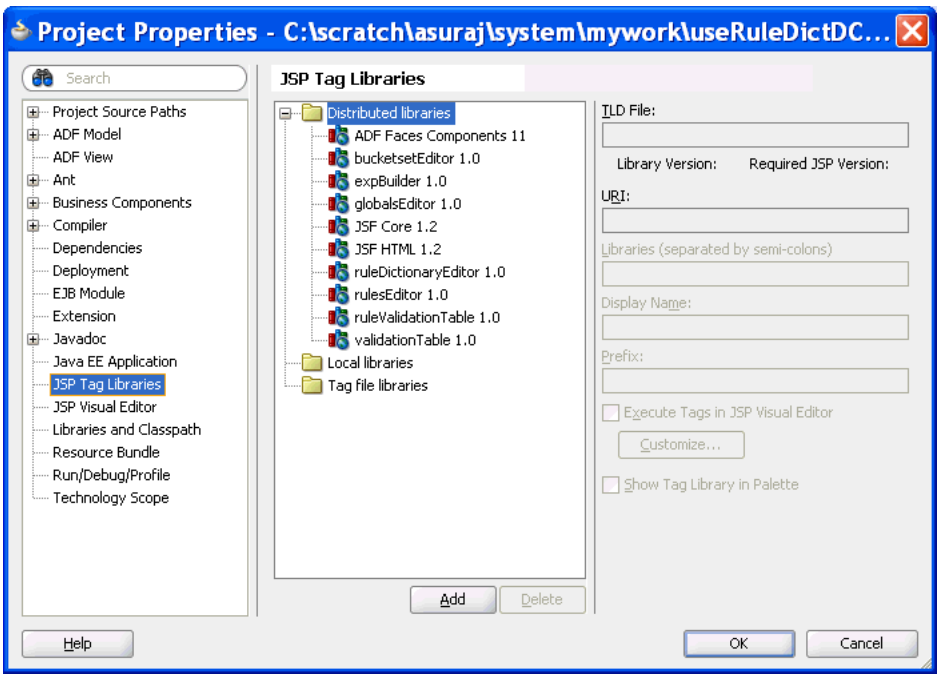
e. Click **OK** to close the Project Properties dialog box.

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



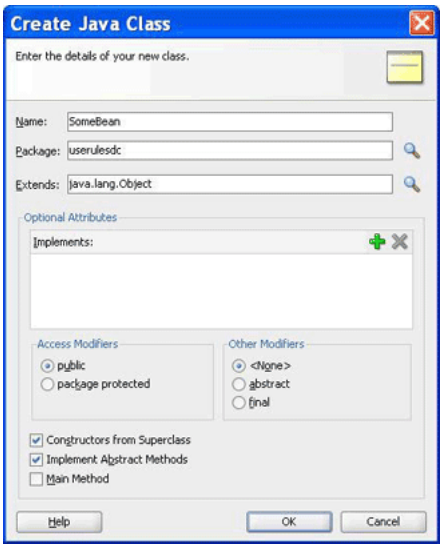
To create the RuleDictionaryModel object:

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.

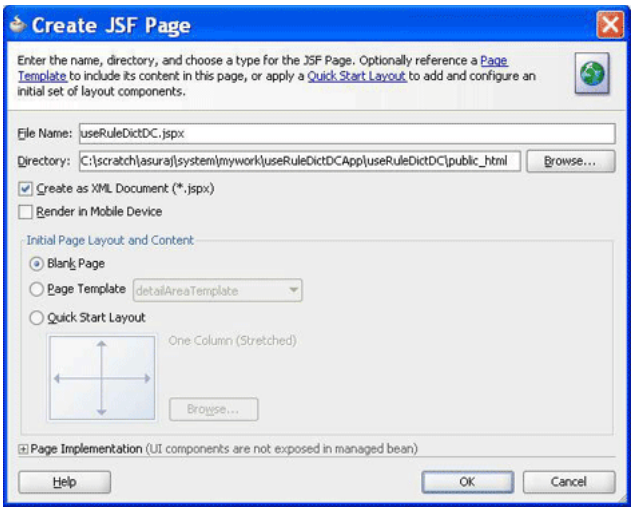
To create the .jspx file for the Rules Dictionary Editor Component tag:

The next task is to create the .jspx file to include the Rules Dictionary Editor Component tag.

The steps are:

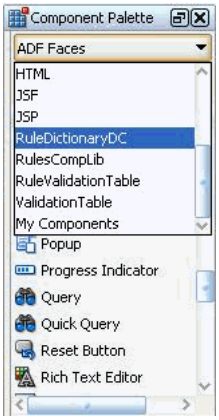
1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** to display the **New Gallery** dialog box.
3. In the **New Gallery** dialog box, select **JSF** under Web Tier from the Categories panel.
4. Select **JSF Page** under Items and click **OK** to display the **Create JSF Page** dialog box.
5. In the **Create JSF Page** dialog box, enter `useRuleDictDC.jspx` 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:

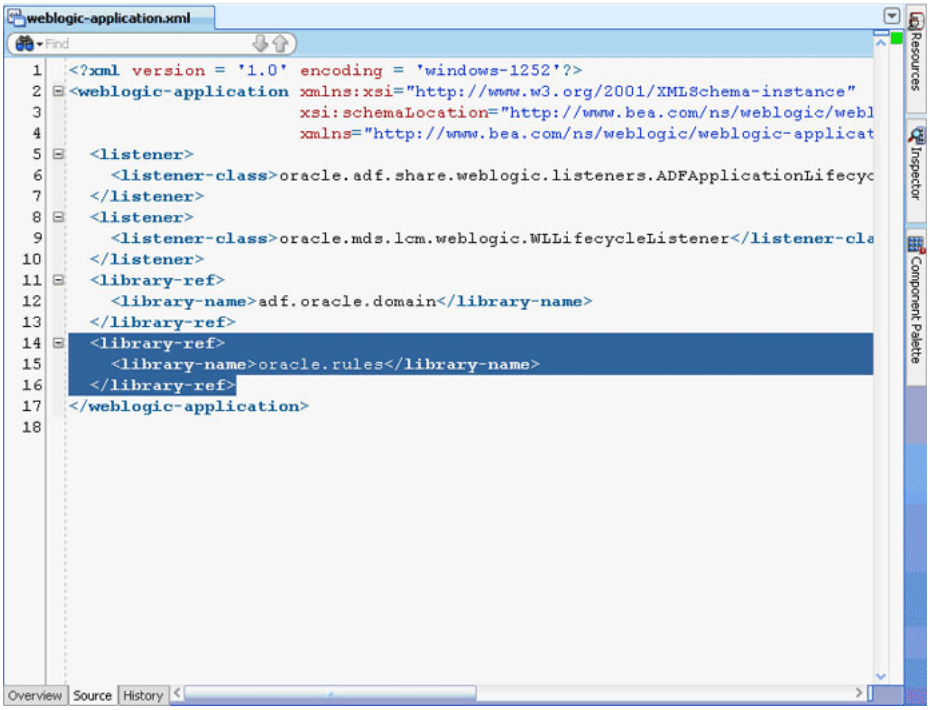
After creating the .jspx file, you must refer to the `oracle.rules` and `oracle.soa.rules_editor_dc.webapp` shared libraries from the weblogic-application.xml file.

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.

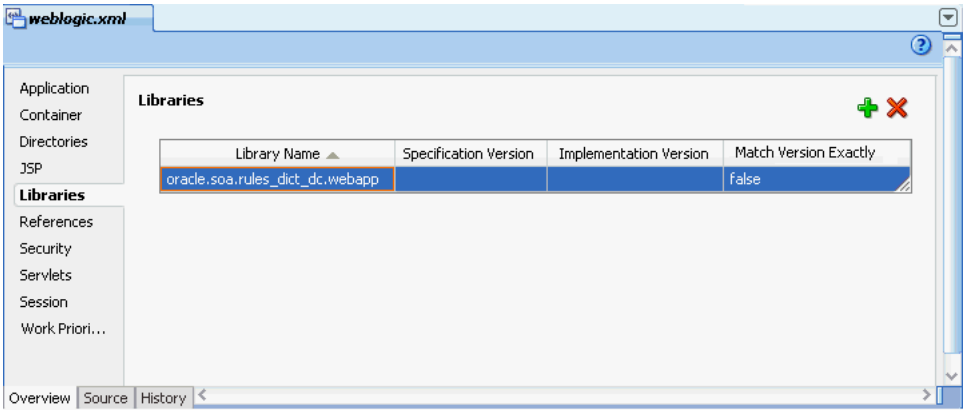
   b. In the New Gallery dialog box, select **Deployment Descriptors** under **General** from the Categories panel.

   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**.

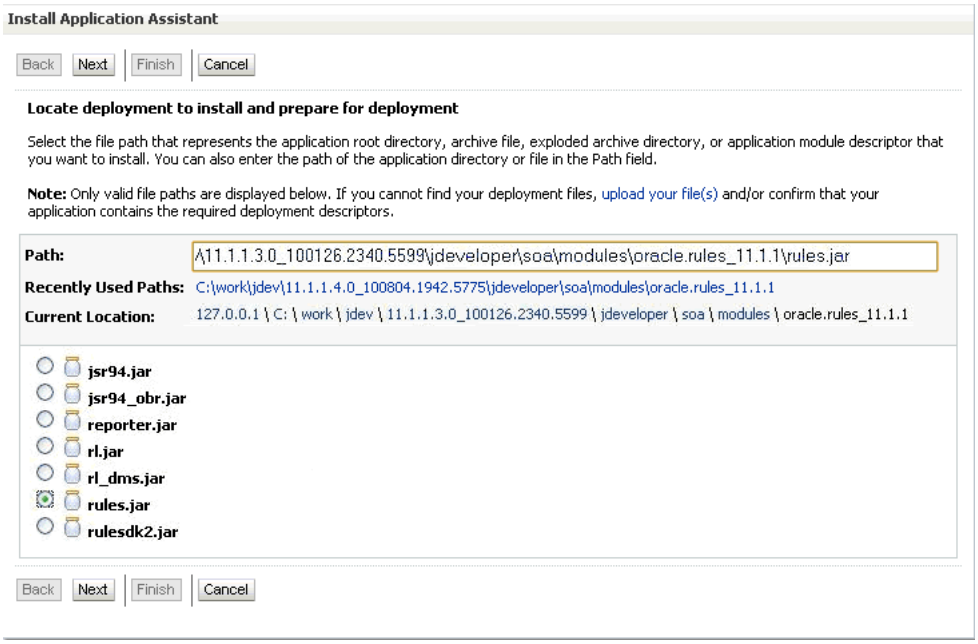
5. Deploy the `oracle.rules` shared library to the embedded Weblogic server:

   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



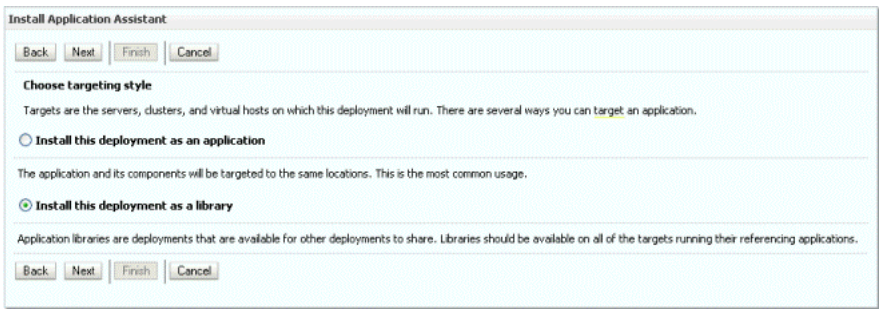
6. Deploy the `oracle.soa.rules_dict_dc.webapp` shared library to the Weblogic server:

   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.

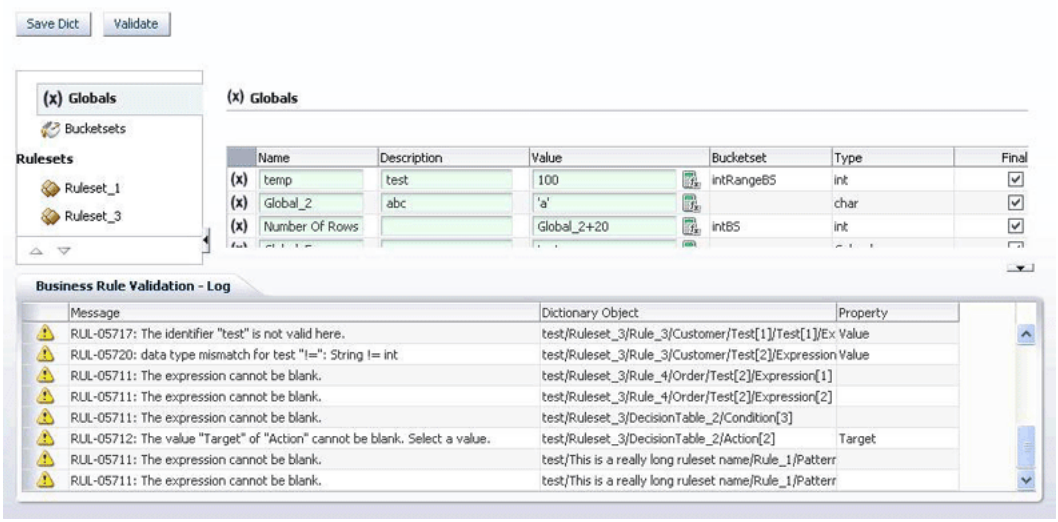
Figure 25–45 oracle.soa.rules_dict_dc.webapp Added to the Deployment List

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



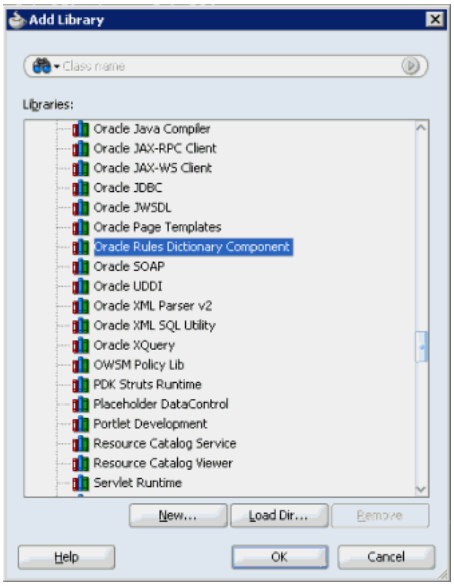
25.3.3 How to Deploy a Rules Dictionary Application to a Standalone Weblogic Server

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 `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.
`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 `displayRuleSetName` 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.

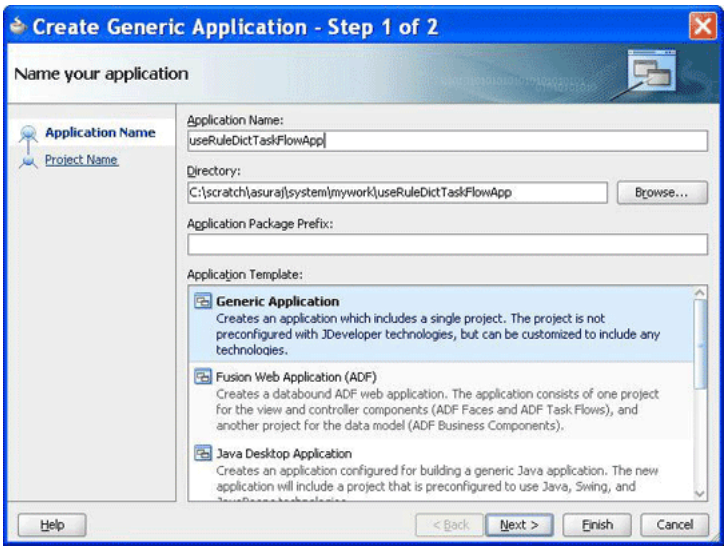
To create a sample application by using the Rules Dictionary Editor task flow:

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.

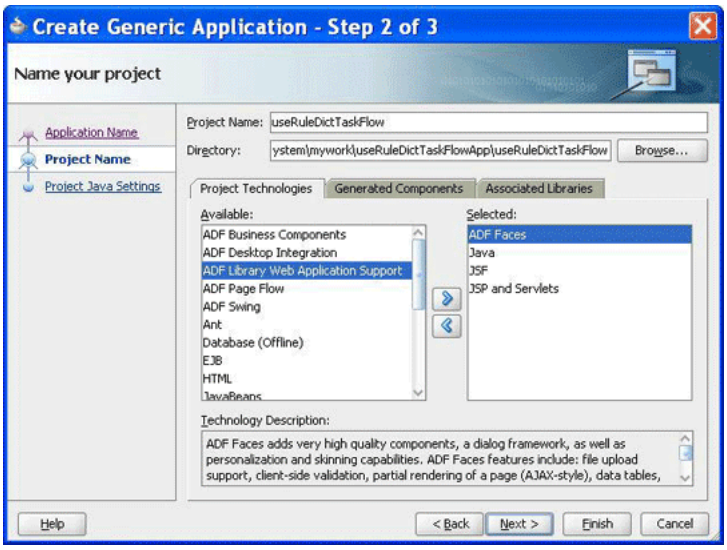
Figure 25–48 Creating a Generic Task Flow Application



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



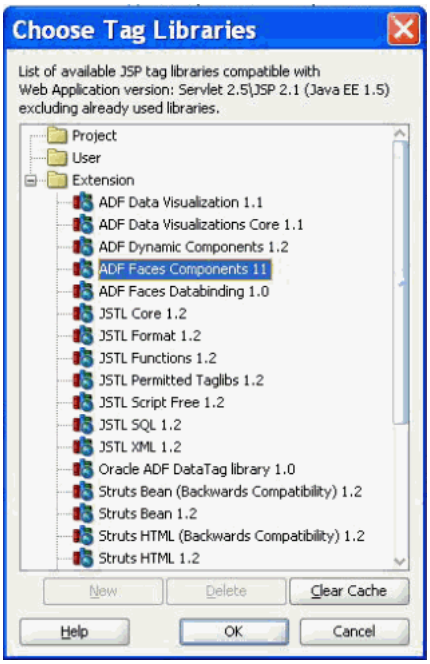
5. Right-click the **useRuleDictTaskFlow** project in the Application Navigator of Oracle JDeveloper, and select **Project Properties** to display the **Project Properties** dialog box.

   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.

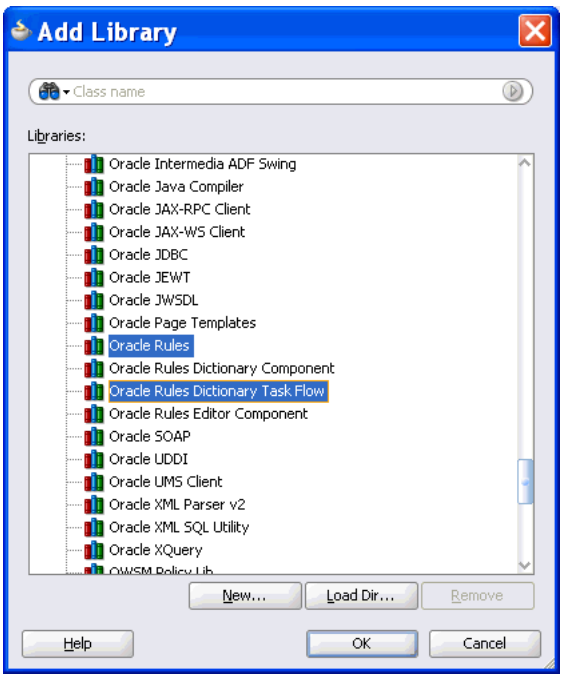
Figure 25–50 Choosing Tab Libraries for the Task Flow Application



   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:

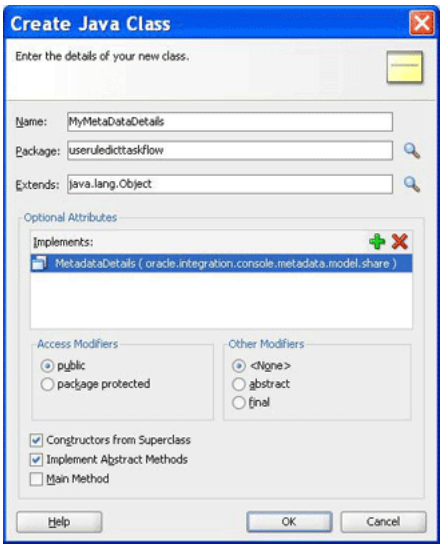
   a. Open Oracle JDeveloper.

   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.NLSPreferences` 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 useruledicttaskflow;

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:

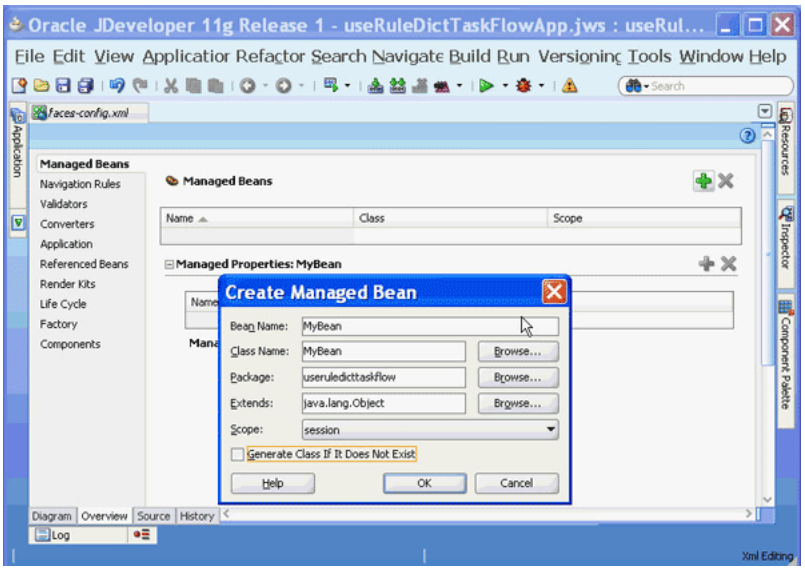
```
package useruledicttaskflow;

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



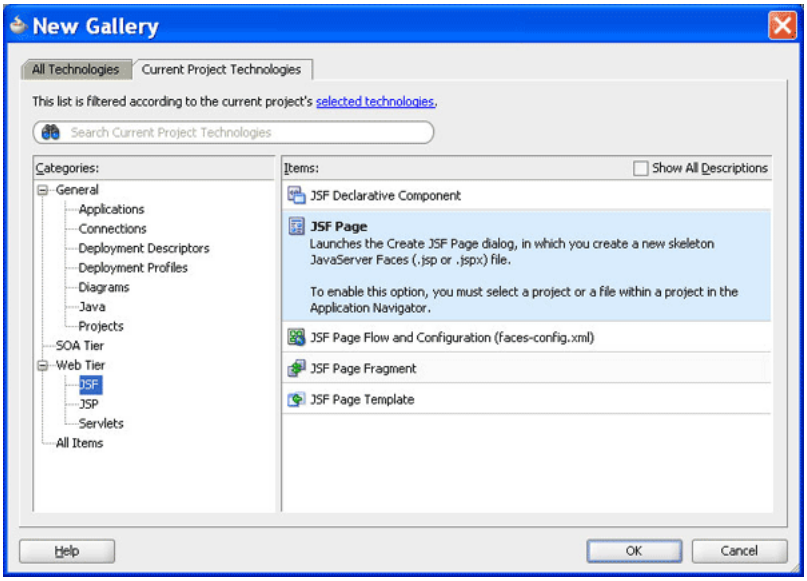
To add a Rules Dictionary Editor task flow in a .jspx file:

The next task is to create the .jspx file to include the Rules Dictionary Editor component tag.

The steps are:

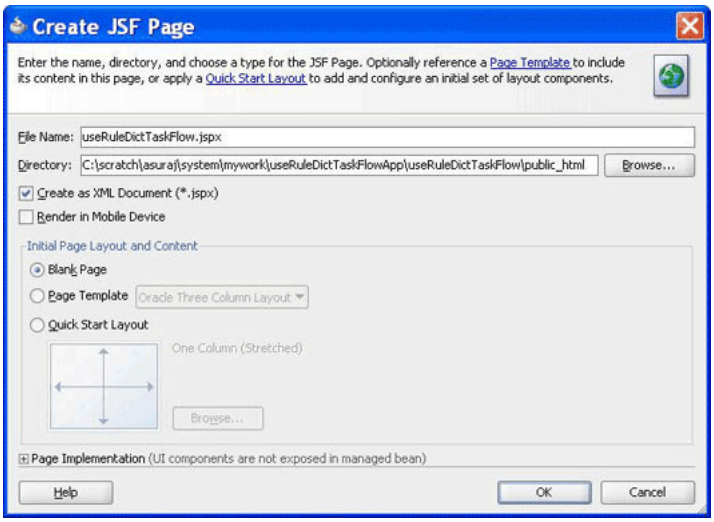
1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** to display the **New Gallery** dialog box.
3. In the **New Gallery** dialog box, select **JSF** under Web Tier from the Categories panel.
4. Select **JSF Page** under Items and click **OK** to display the **Create JSF Page** dialog box as shown in Figure 25–54.

Figure 25–54 Creating the JSF Page File to Include the Rules Dictionary Editor Task Flow



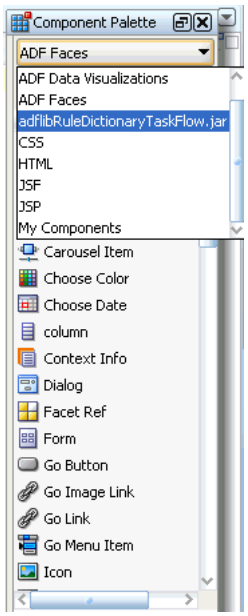
5. In the **Create JSF Page** dialog box, enter `useRuleDictTaskFlow.jspx` as the file name as shown in Figure 25–55.

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.

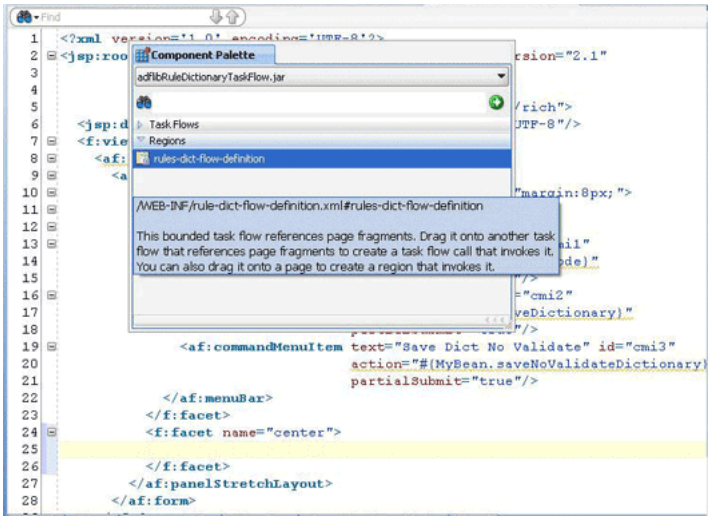
Figure 25–56 Rules Dictionary Task Flow JAR in the Component Palette



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.jspx` file with the task flow added:

```
<f:view>
    <af:document id="d1">
      <af:form id="f1">
        <af:panelStretchLayout id="psl1" 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"/>
```

```
        <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.

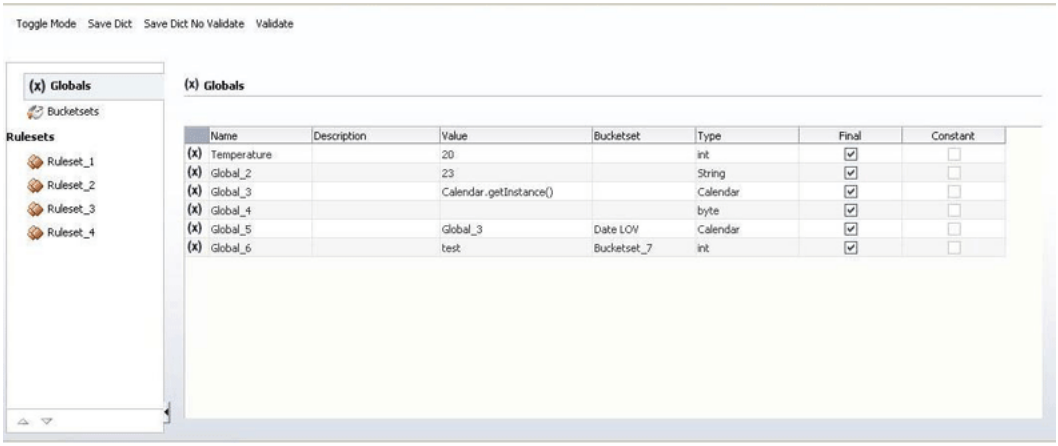
For more information on referring to the shared libraries, see Section 25.3.2, "How to Create and Run a Sample Application by Using the Rules Dictionary Editor Component."

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:

26

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:

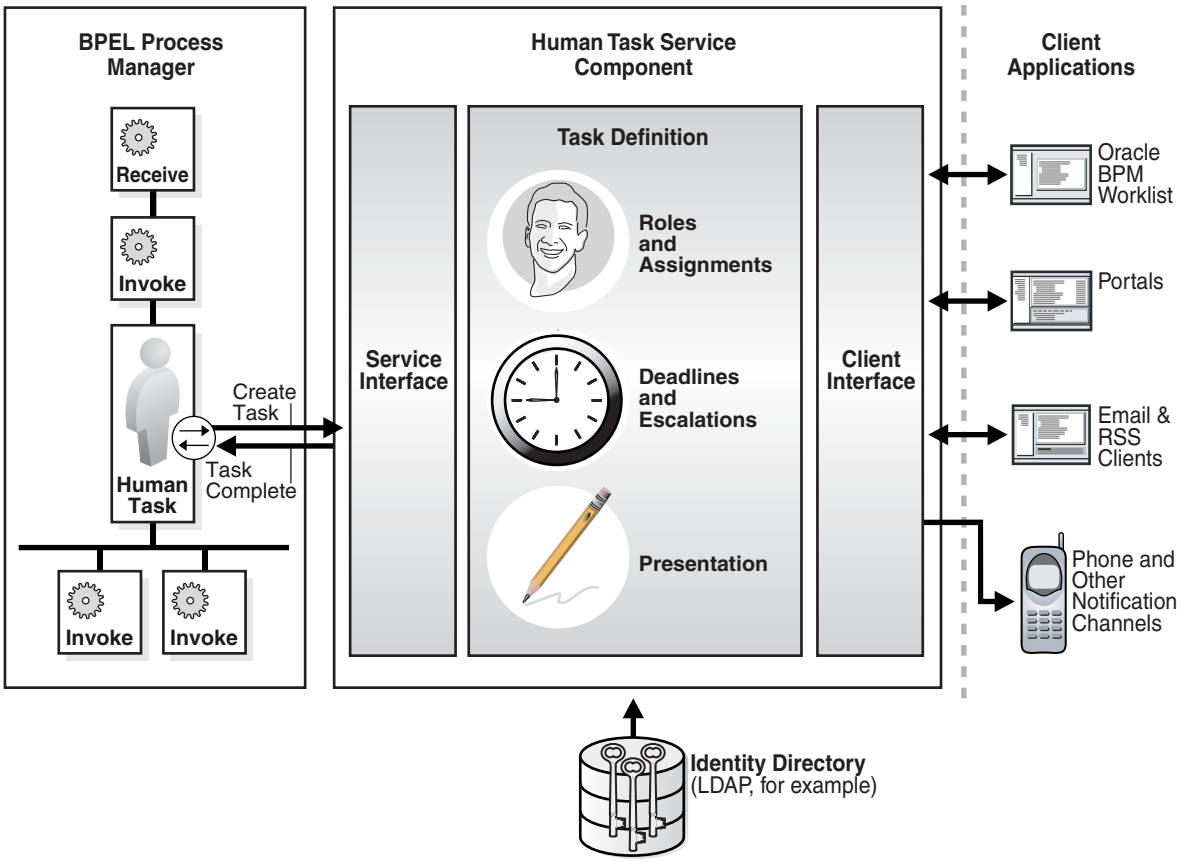
26.1 Introduction to Human Workflow

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

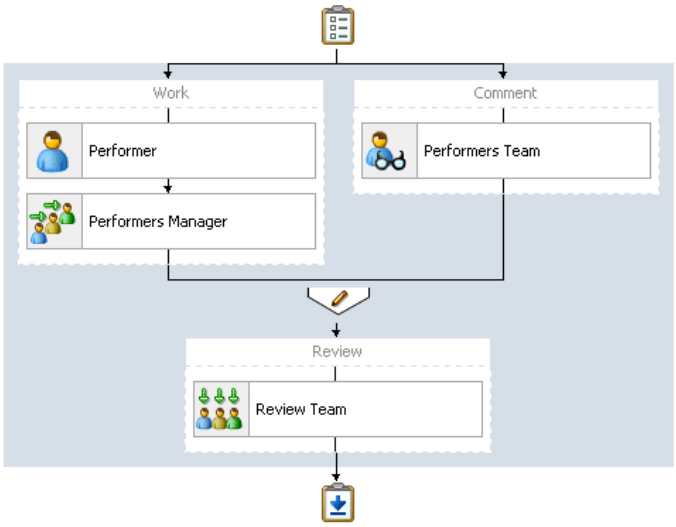
Before designing a human task, it is important to understand the design and runtime concepts. A typical task consists of a subject, priority, task participants, task parameters or data, deadlines, notifications or reminders, and task forms. This section provides an overview of key concepts.

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 `jlondon` or `jstein` 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 `jcooper` and `fkafka` 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.

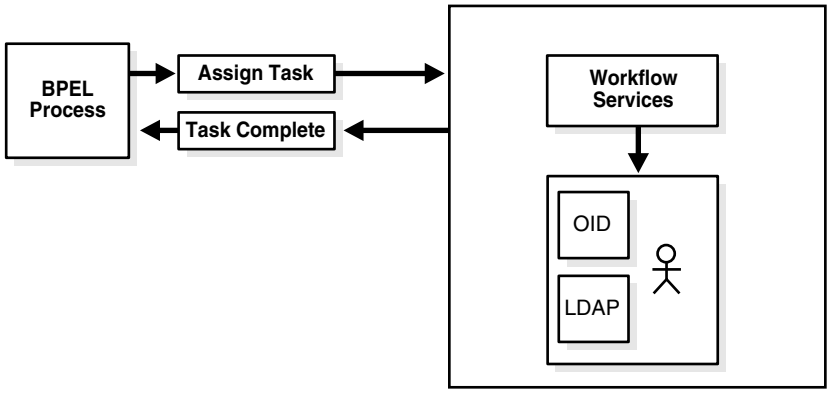
26.3.1 Human Workflow Use Cases

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.

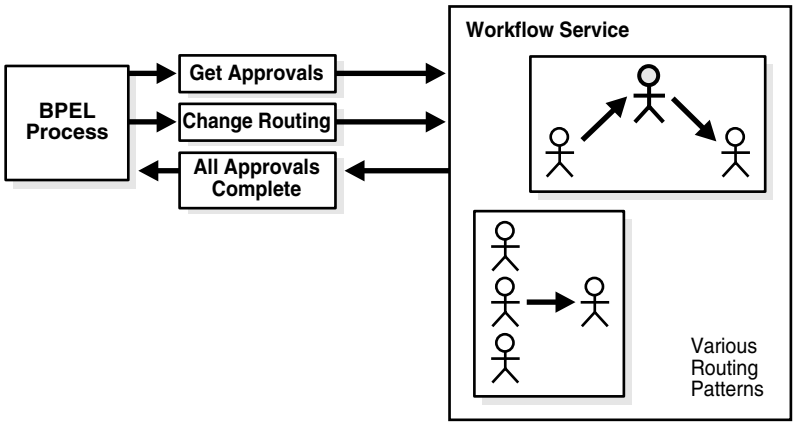
Figure 26–3 Assigning Tasks to a User or Role from a Directory



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 that $100,000. This scenario is shown in Figure 26–4.

Figure 26–4 Flow Patterns and Routing Policies

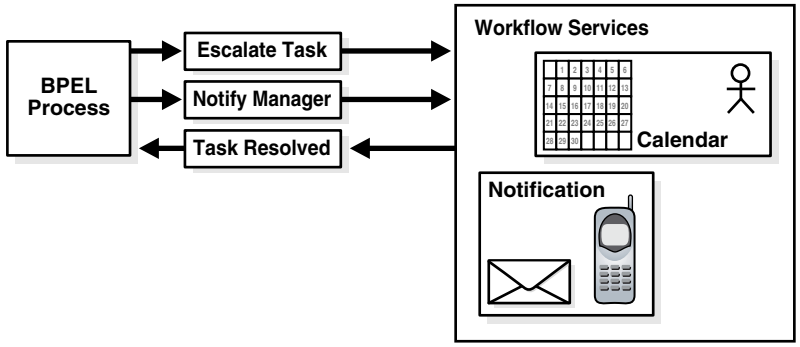


You can use these types as building blocks to create complex workflows.

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>
 <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

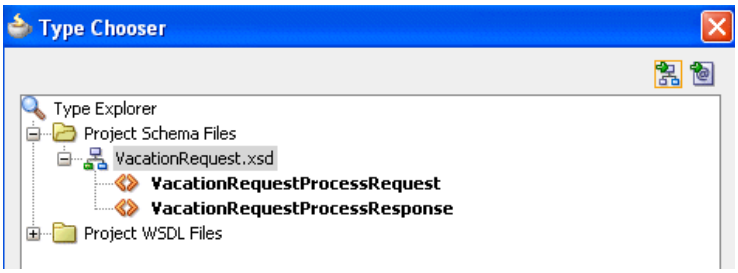
To create an application and a project with a BPEL process:

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**.

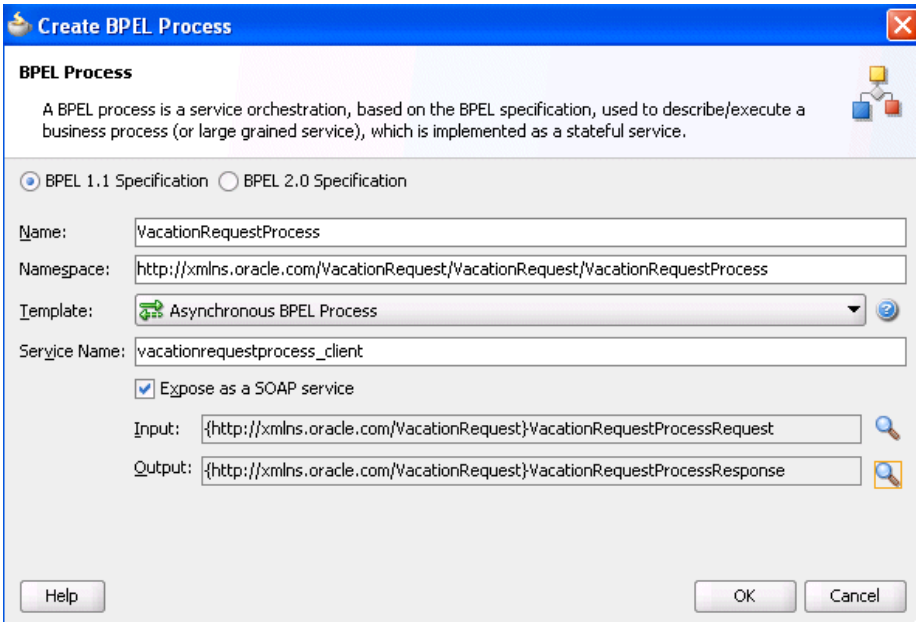
    You are returned to the Create BPEL Process dialog.

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.

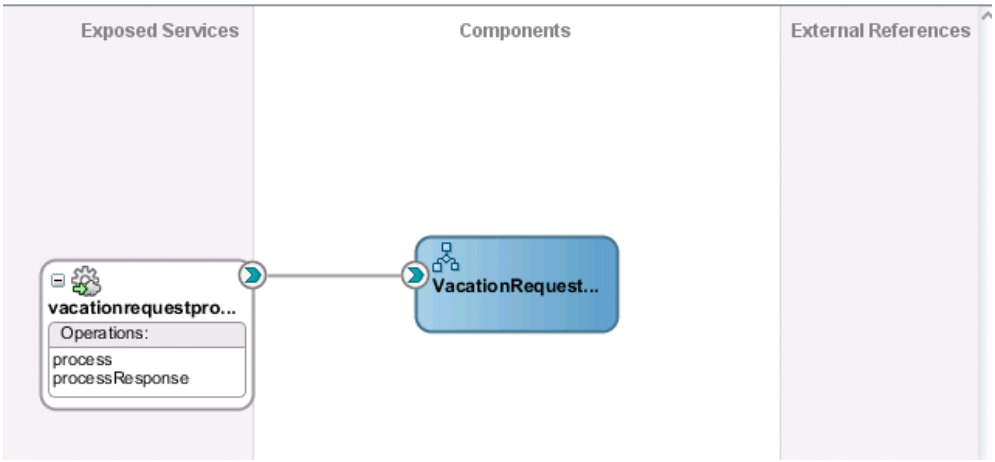
Figure 26–7 BPEL Process Dialog



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.

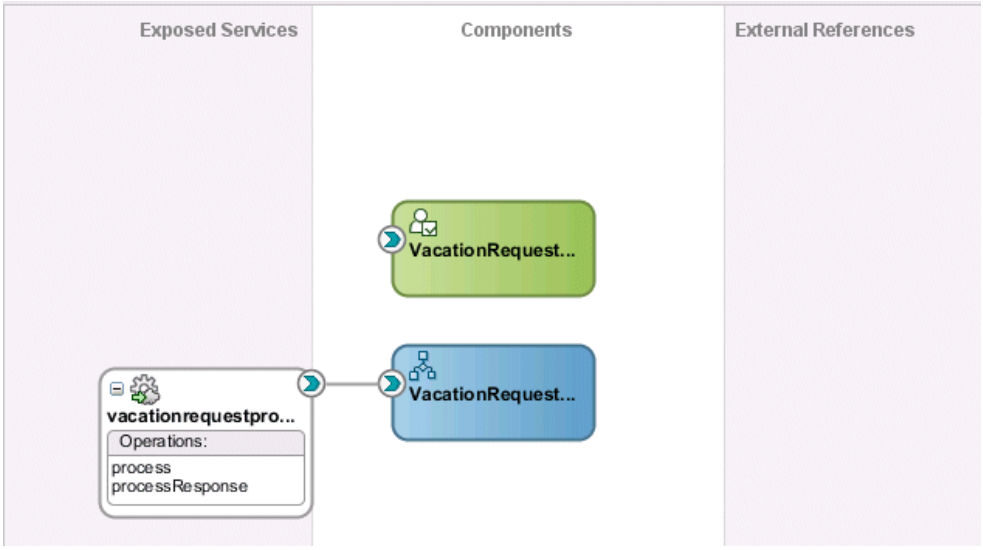
Table 26–2 Create Human Task Dialog Fields and Values

Field	Value
Name	Enter `VacationRequestTask`.
Namespace	Accept the default value.
Create Composite Service with SOAP Bindings	Do *not* 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

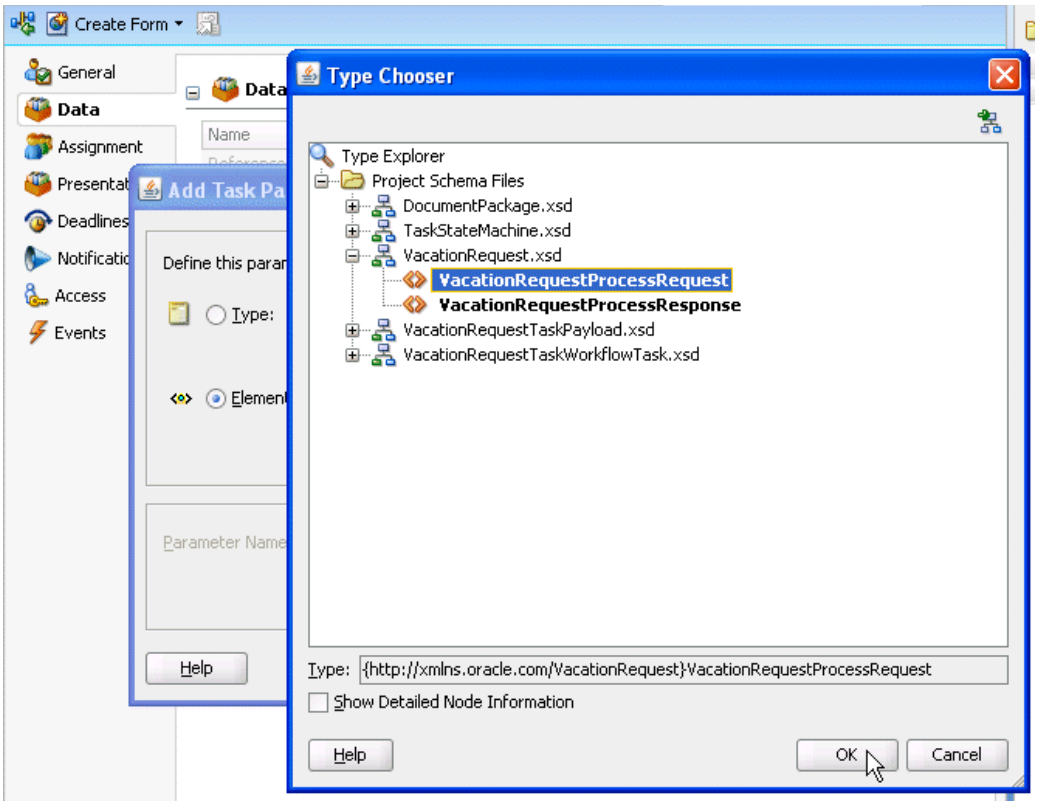
To design the human task:

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.

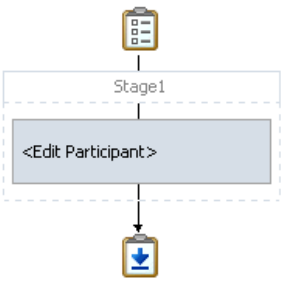
   The Type Chooser dialog appears.
8. Expand and select **Project Schema Files** > **VacationRequest.xsd** > **VacationRequestProcessRequest**, and click **OK**. Figure 26–10 provides details.

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.

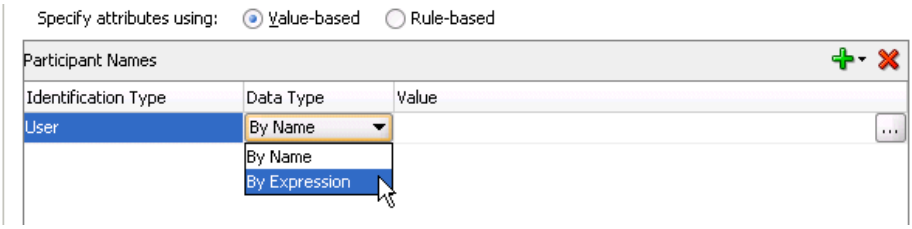
    The Edit Participant Type dialog appears. You now add participants to this task. As described in Section 26.2.1.1.2, "Participant Type," Oracle SOA Suite provides several out-of-the-box patterns known as participant types for addressing specific business needs.

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

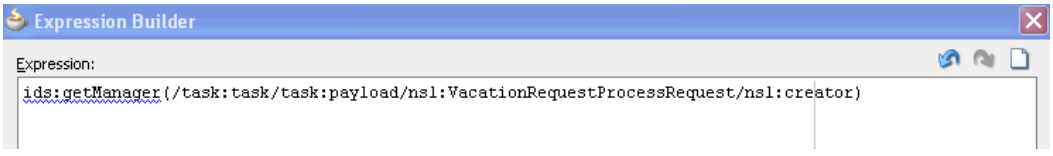


17. In the **Value** column, click the **Browse** icon (the dots) to invoke the Expression Builder dialog.
18. In the dropdown list in the **Functions** section, select **Identity Service Functions**.
19. Select **getManager**. This function gets the manager of the user who created the vacation request 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:

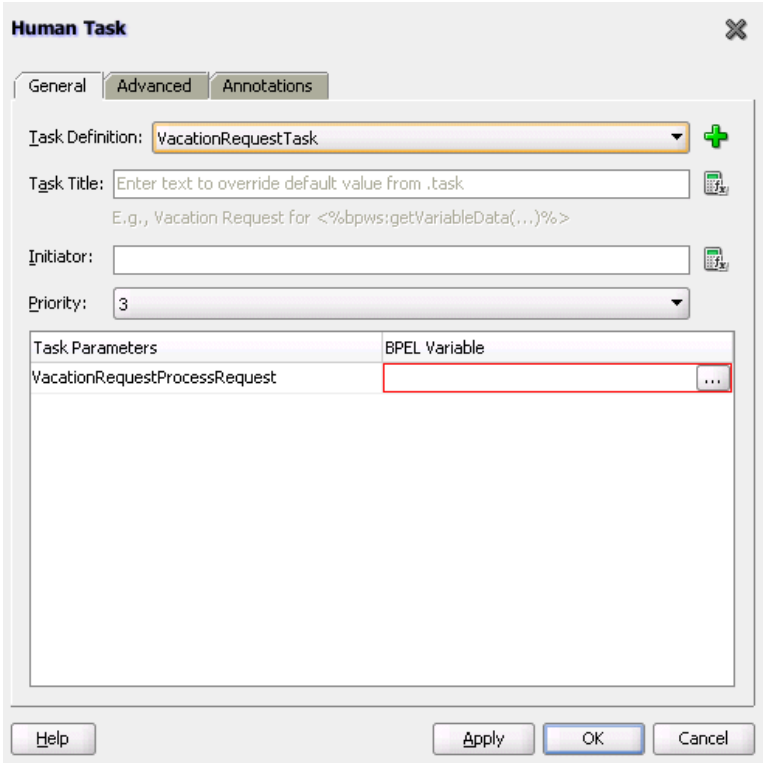
1. In the Application Navigator, double-click **composite.xml**.
2. Double-click the **VacationRequestProcess** BPEL process service component in the SOA Composite Editor.

   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).

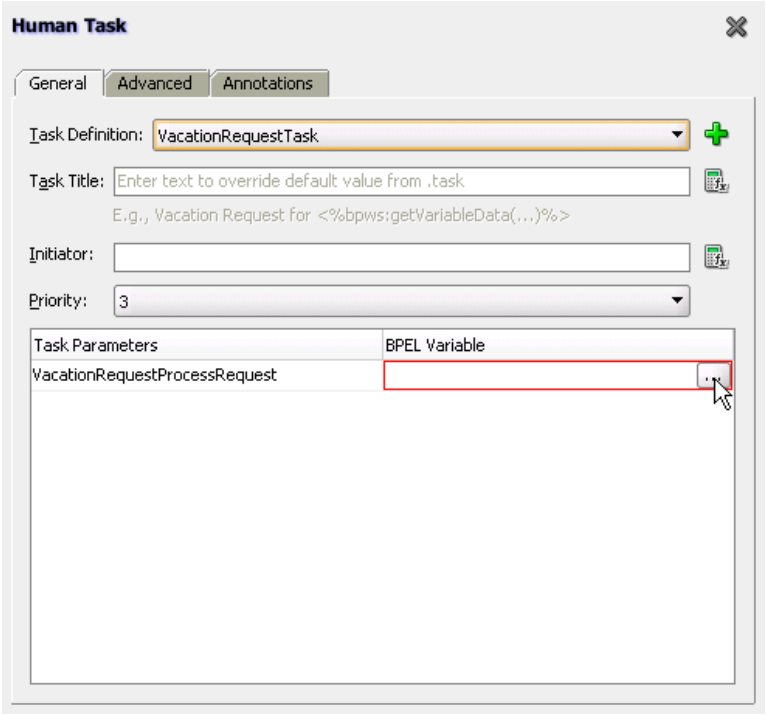
   The dialog refreshes as shown in Figure 26–14 to display additional fields.

Figure 26–14 Human Task Dialog



7. In the **BPEL Variable** column, click the **Browse** icon (dots) shown in Figure 26–15.

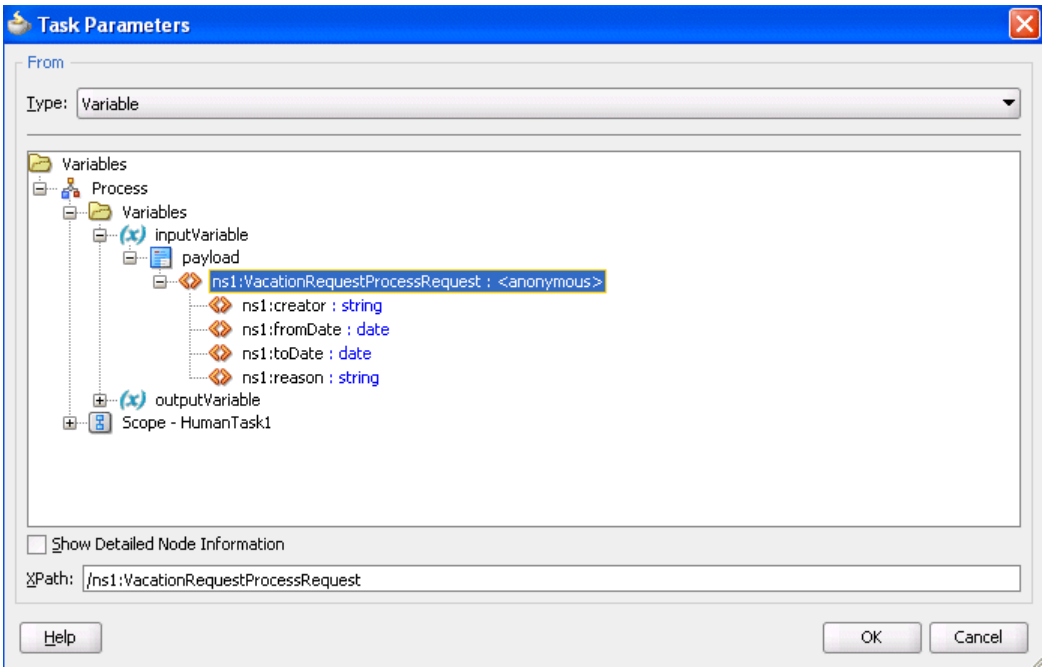
Figure 26–15 BPEL Variable Entry



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.

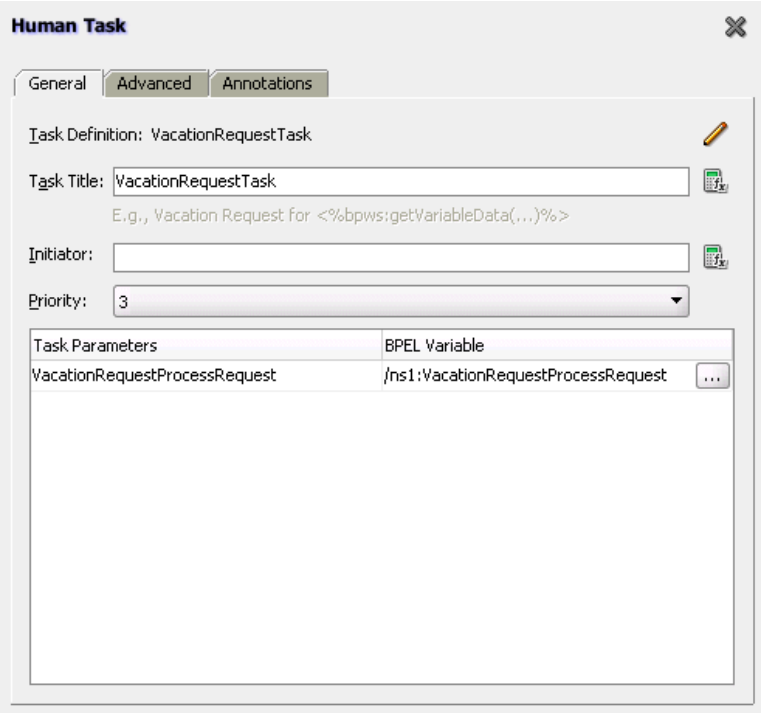
Figure 26–16 Variable Selection



10. Click **OK**.

When complete, the dialog looks as shown in Figure 26–17:

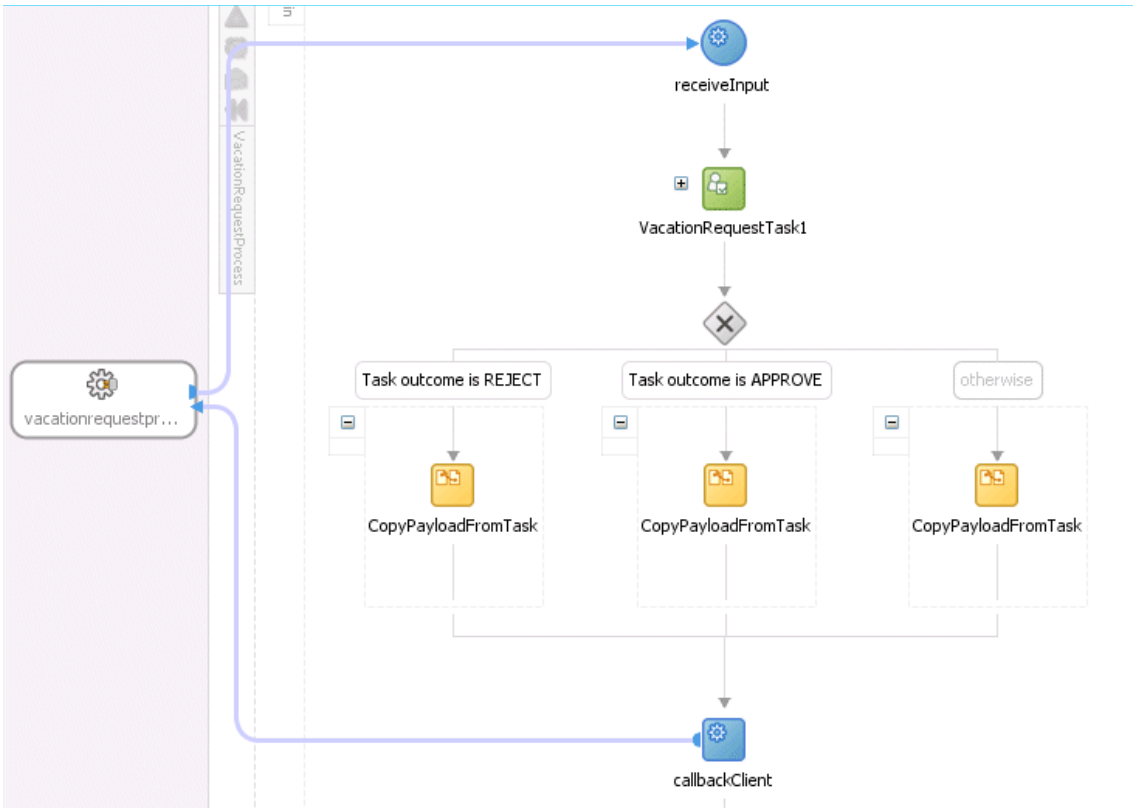
Figure 26–17 BPEL Variable



11. Click **OK** to close the Human Task dialog.

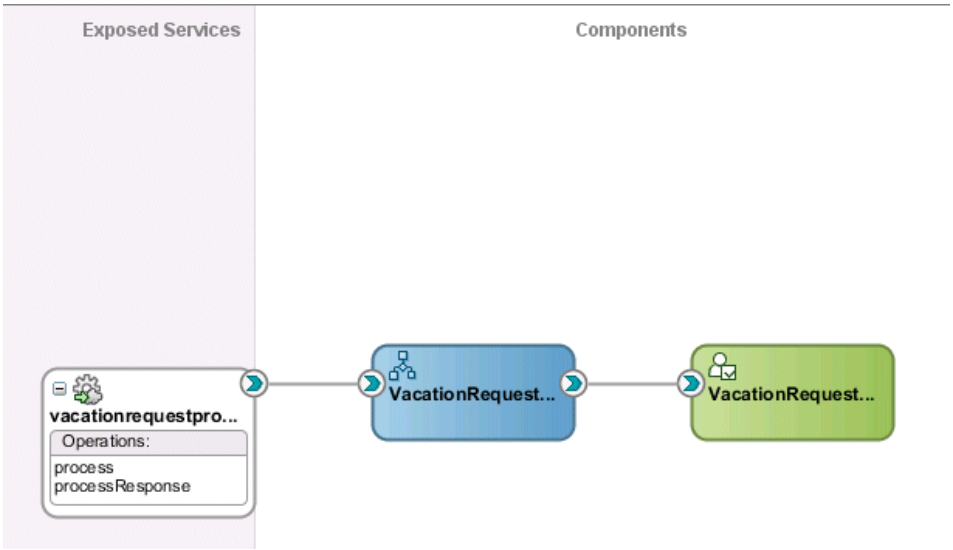
    The human task activity appears as shown in Figure 26–18.

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

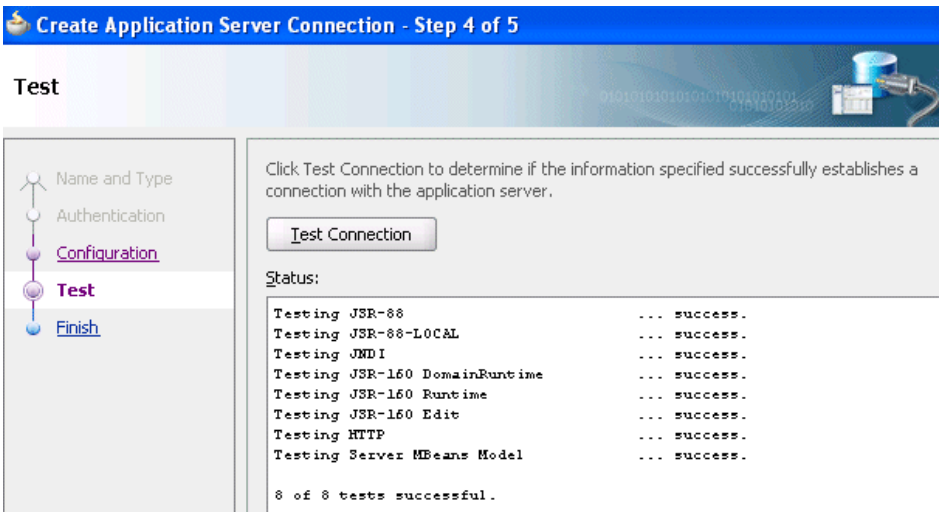
You are now ready to create a connection to the application server on which Oracle SOA Suite is installed and configured with the SOA Infrastructure. These instructions describe how to create a connection to Oracle WebLogic Server. For information about creating a connection to other application servers 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

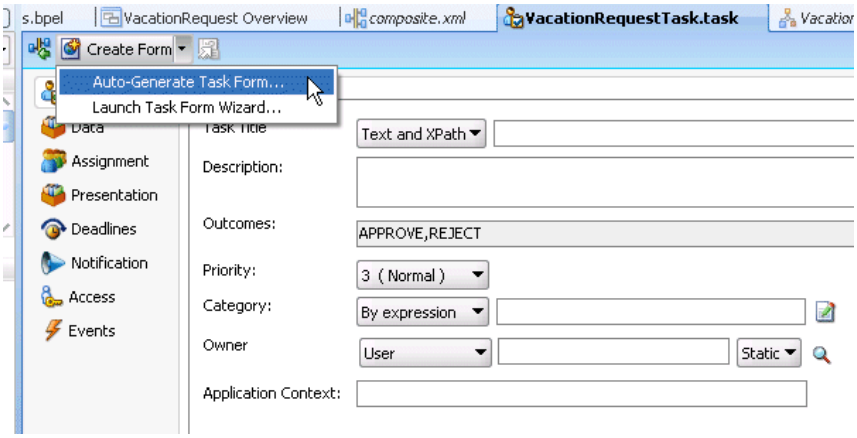
You are now ready to create a project for the task form. This is a separate project from the one in which you created the human task.

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 11*g*, 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 WebLogic Server 11*g* 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

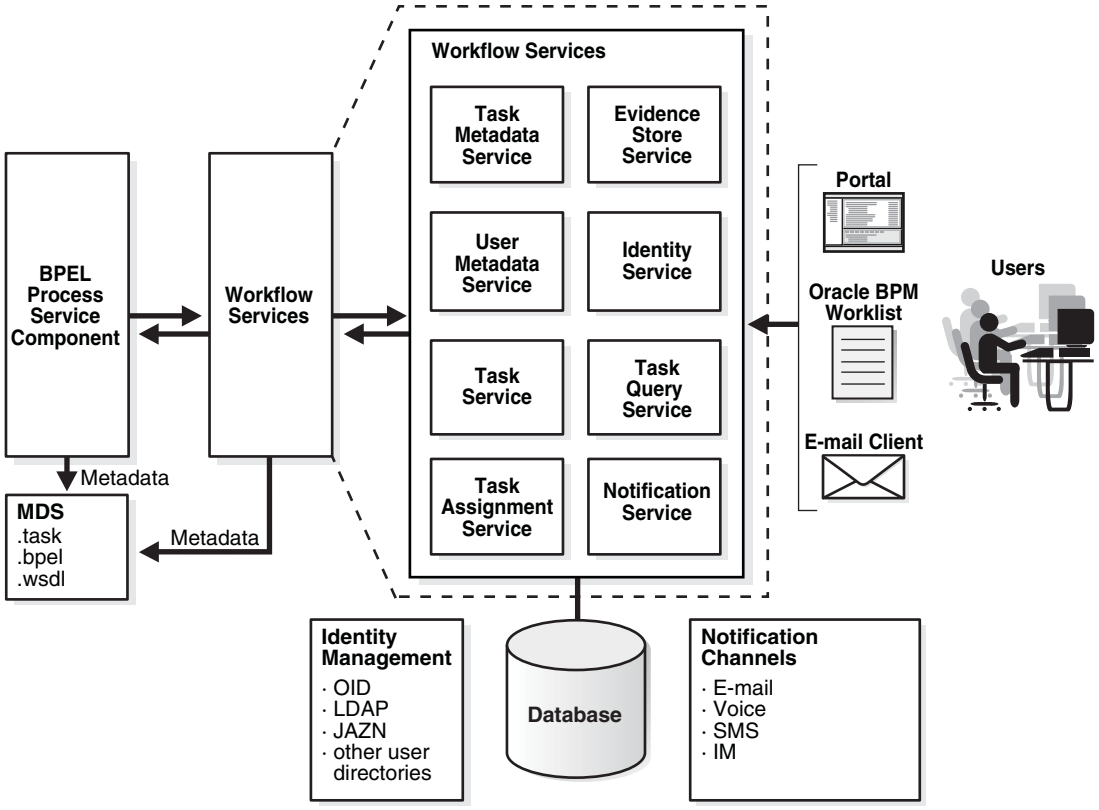
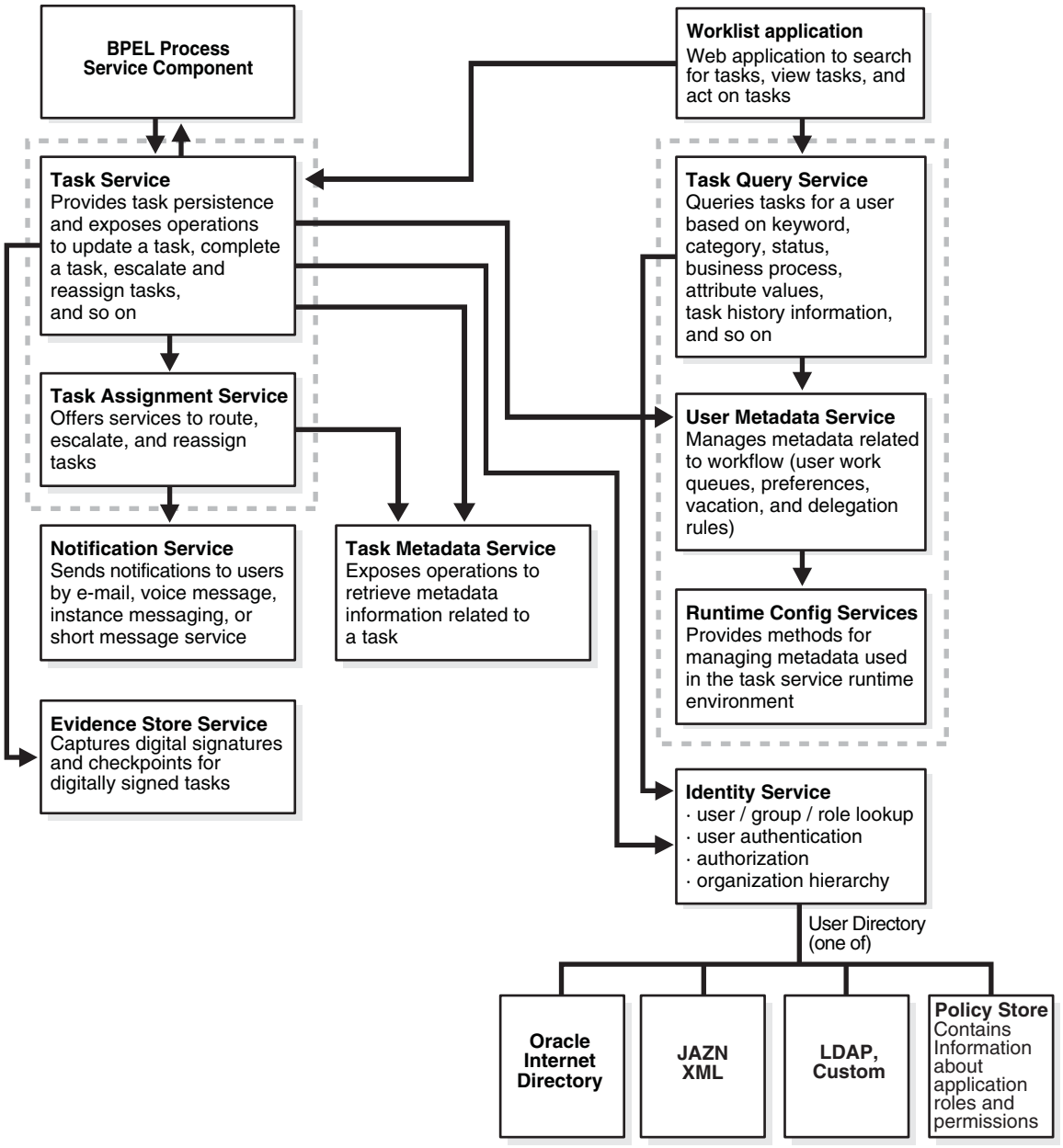


Figure 26–23 shows the interactions between the services and the business process.

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*.

27

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.

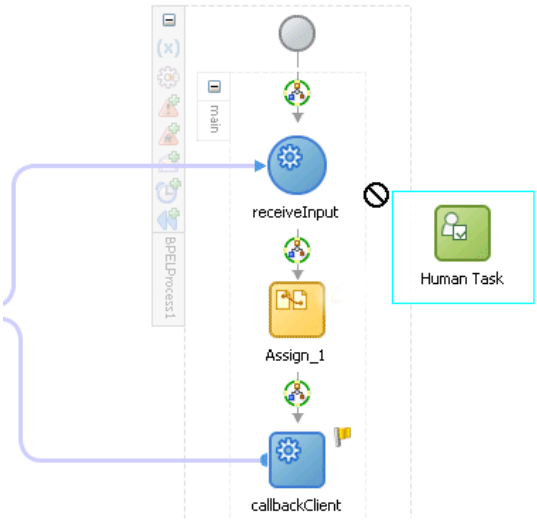
The Human Task Editor enables you to specify human task metadata, such as task outcome, payload structure, assignment and routing policy, expiration and escalation policy, notification settings, and so on. This information is saved to a metadata task configuration file with a `.task` extension. In addition, 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.

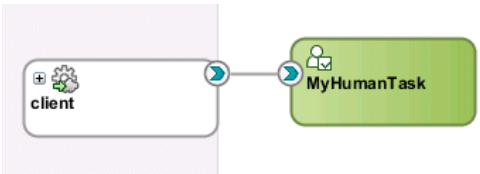
   The Create Human Task dialog appears.
4. In the **Name** field, enter a name.

   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:

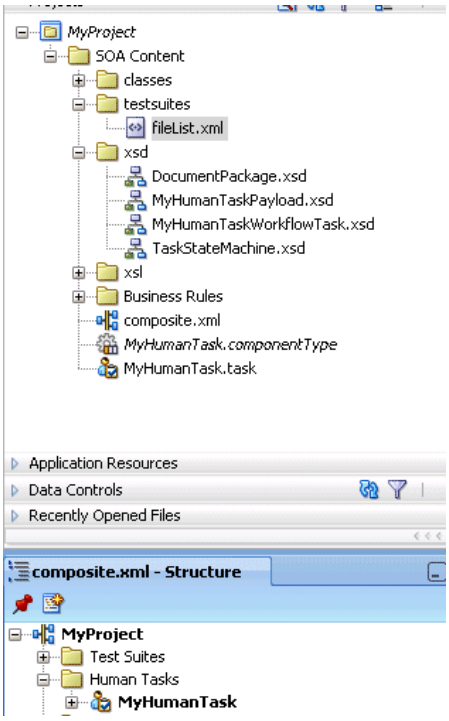
- The human task settings specified in the Human Task Editor are saved to a metadata task configuration file in the metadata service (MDS) repository with a `.task` extension. This file appears in the Application Navigator under ***SOA_Project_Name*** **> SOA Content**. You can re-edit the settings in 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."

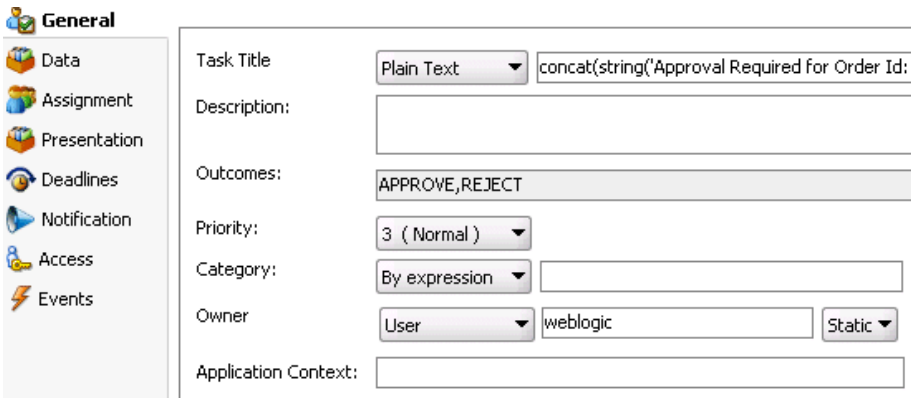
27.3.3 How to Access the Sections of the Human Task Editor

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

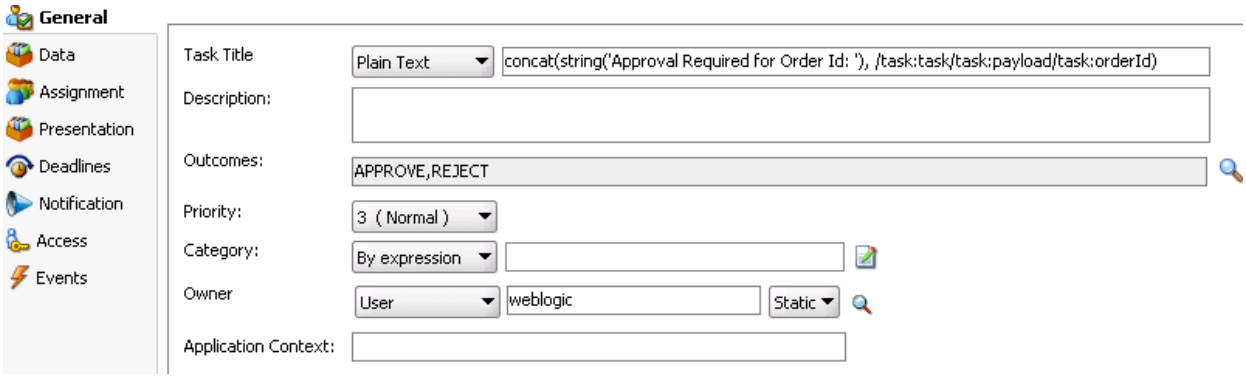
To specify the title, description, outcome, priority, category, owner, and application context:

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



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.

1. 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.

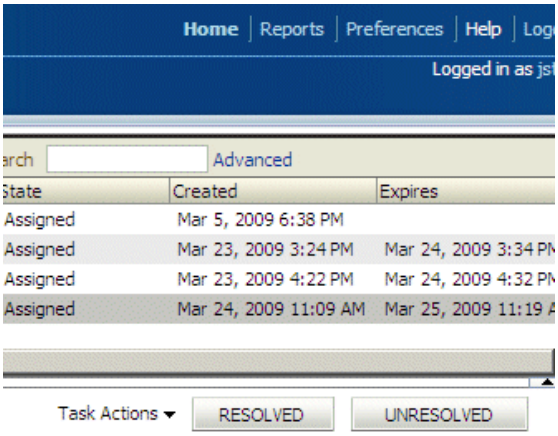
27.3.4.2 Specifying a Task Description

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

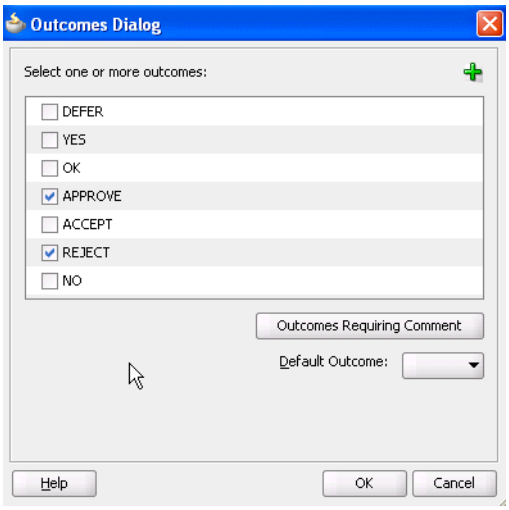
The task outcomes can also have runtime display values that are different from the actual outcome value specified here. This permits outcomes to be displayed in a different language in Oracle BPM Worklist. For more information about internationalization, see Section 27.3.8.2, "Specifying Multilingual 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 `Delete`). Specifying the same name can cause problems at runtime. ▪ Do not specify a custom name with blank spaces (for example, `On Hold`). 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

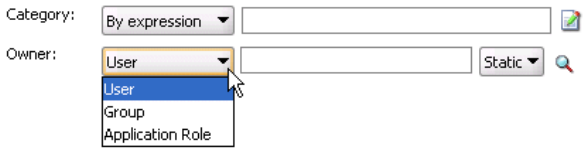
Task owners can be selected by browsing the user directory (Oracle Internet Directory, Java AuthoriZatioN (JAZN)/XML, LDAP, and so on) or a list of application roles configured for use with Oracle SOA Suite.

To specify a task owner statically through the user directory or a list of application roles:

1. In the first list to the right of the **Owner** field in the **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



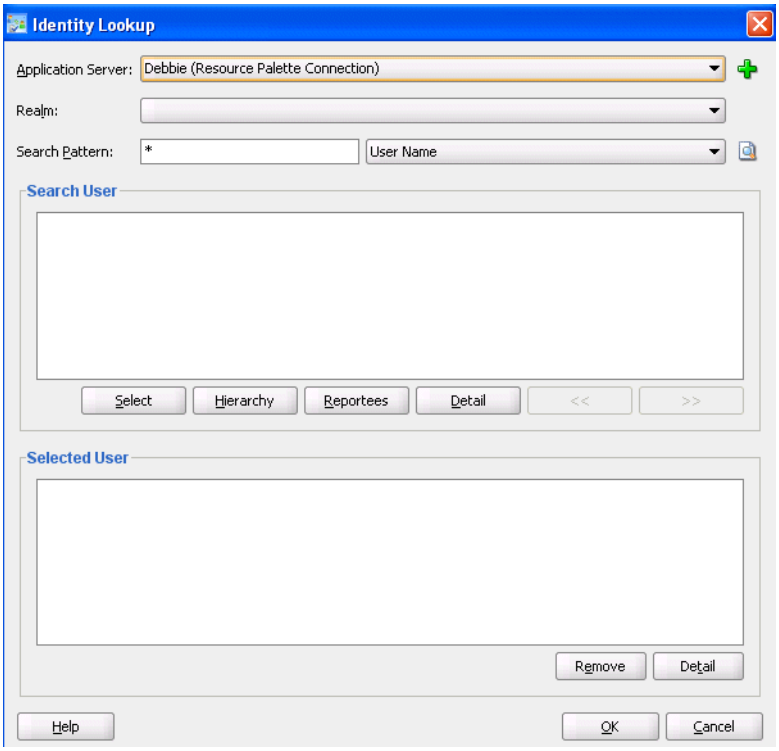
2. In the second list to the right of the **Owner** field in the **General** section, select **Static**.
3. See the step in Table 27–4 based on the type of owner you selected.

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



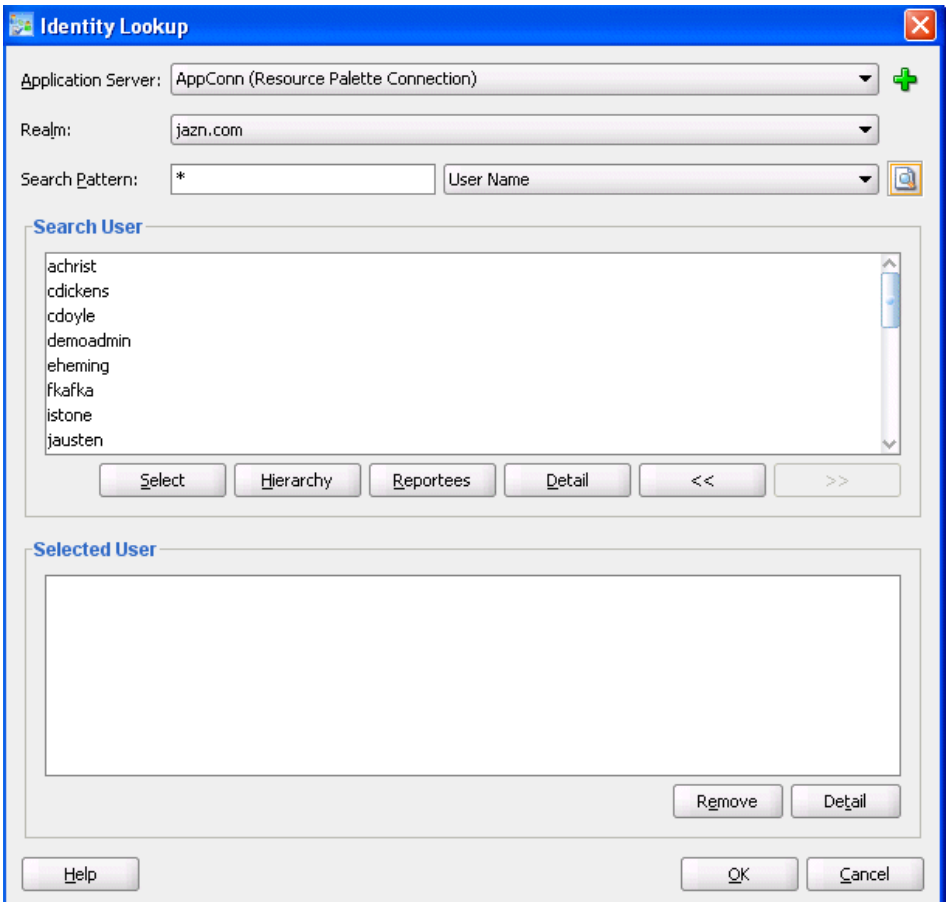
To select a user or group, you must first create an application server connection by clicking the **Add** icon. Note the following restrictions:

- 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).

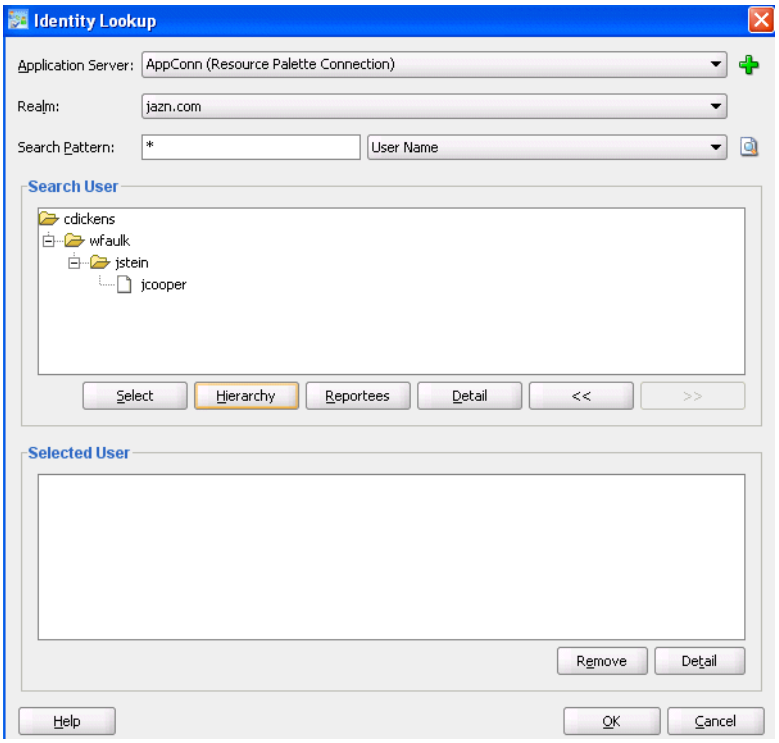
b. Search for the owner by entering a search string such as `jcooper`, `j*`, `*`, and so on. Clicking the **Lookup** icon to the right of the **User Name** field fetches all the users that match the search criteria. Figure 27–11 provides details. One or more users or groups can be highlighted and selected by clicking **Select**.

Figure 27–11 Identity Lookup with Realm Selected



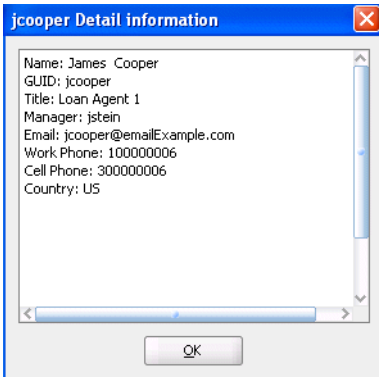
c. View the hierarchy of a user by highlighting the user and clicking **Hierarchy**. Similarly, clicking **Reportees** displays the reportees of a selected user or group. Figure 27–12 provides details.

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.

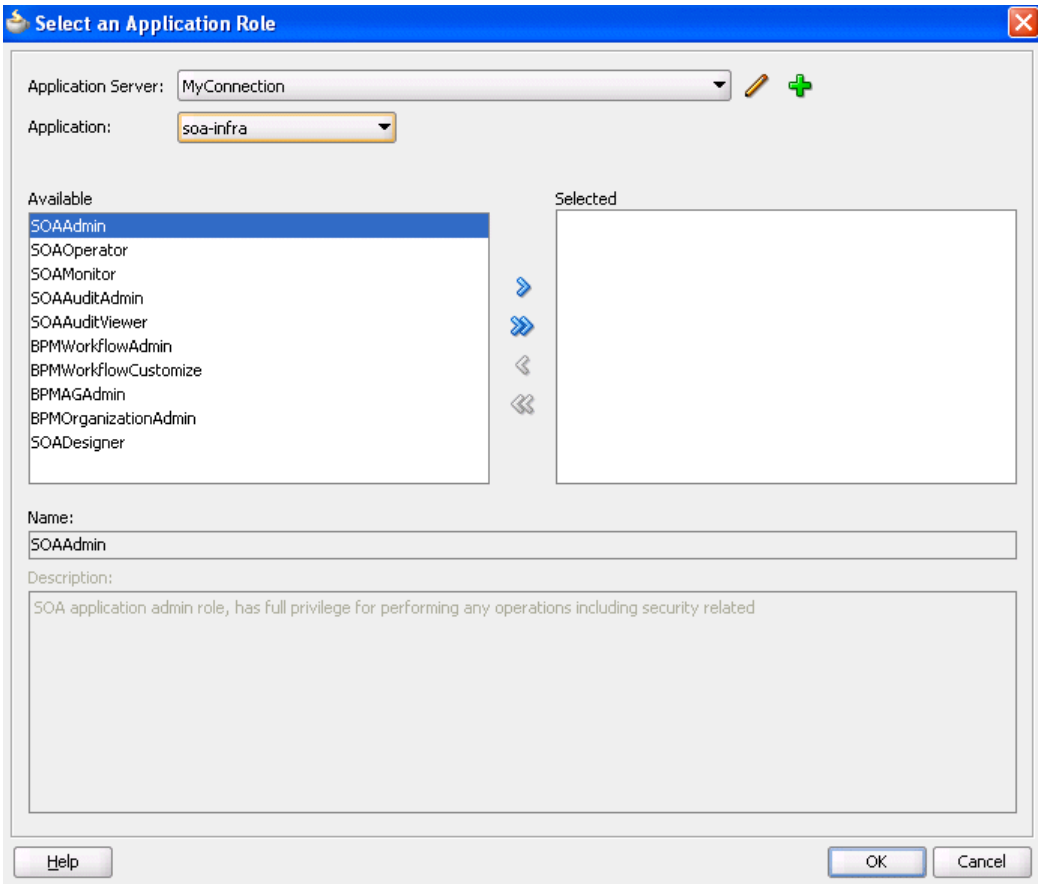
5. If you selected **Application Role**, the Select an Application Role dialog appears.

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**.

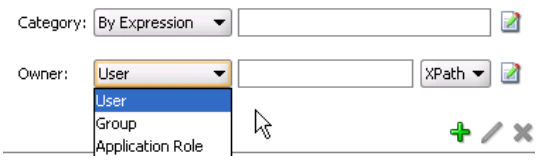
27.3.4.6.2 Specifying a Task Owner Dynamically Through an XPath Expression

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.

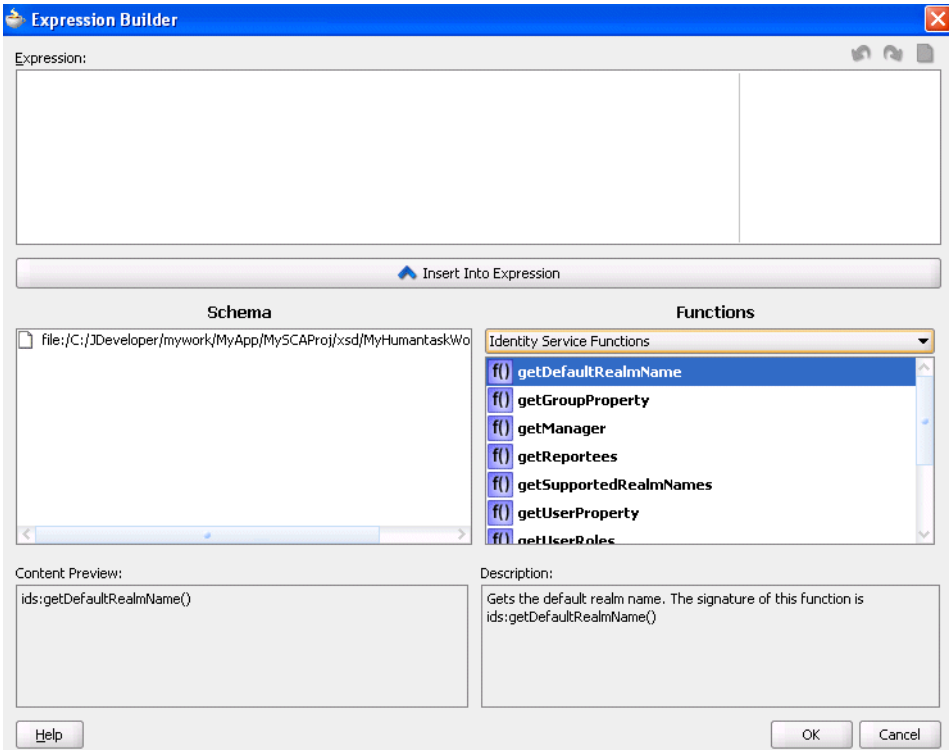
Figure 27–15 Specify a Task Owner Dynamically



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

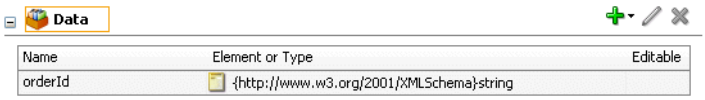
Figure 27–17 shows the **Data** section of the Human Task Editor.

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

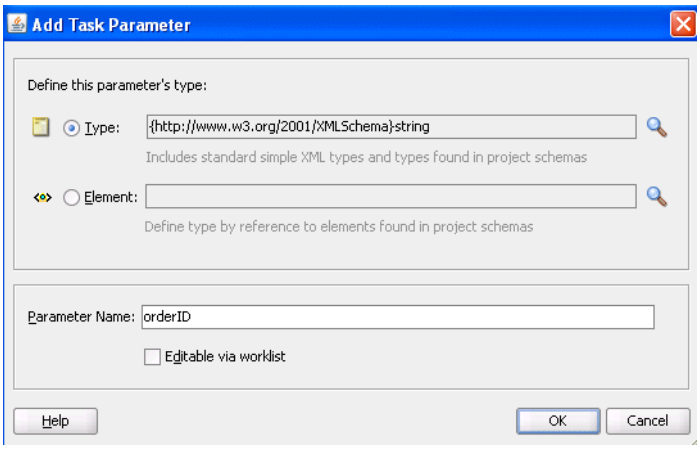


To specify the task payload data structure:

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.

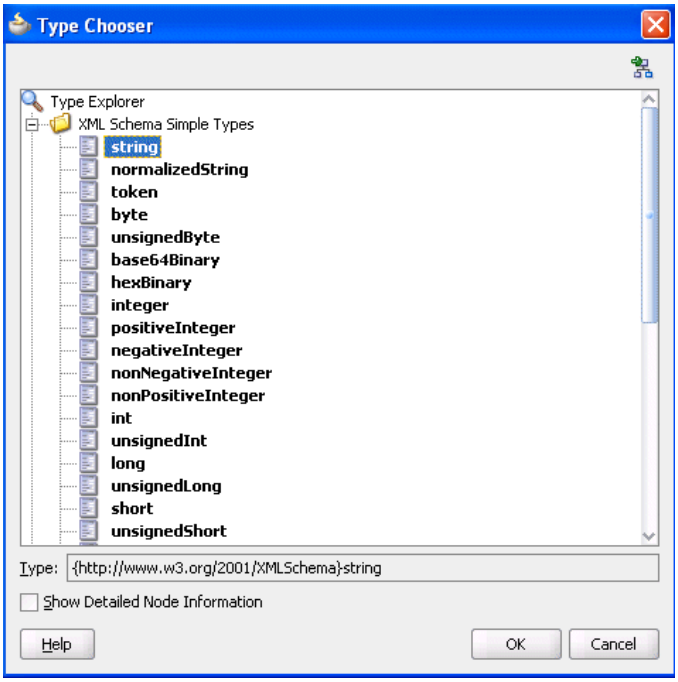
Table 27–5 (Cont.) Add Task Parameter Dialog Fields and Values

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.

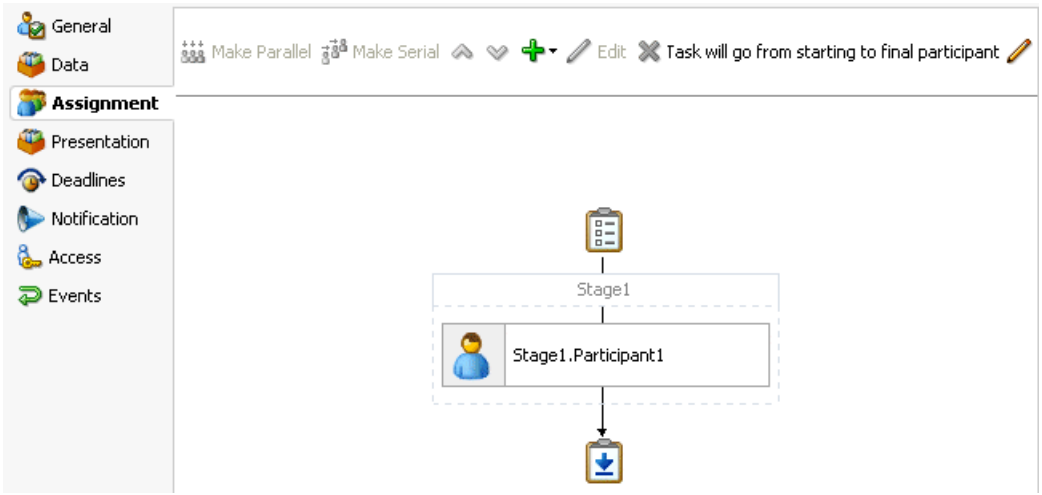
   Your selection displays in the **Data** section.

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."

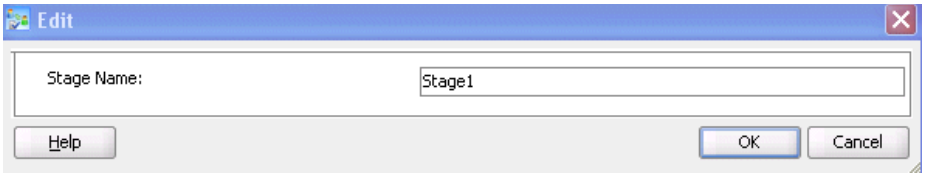
To specify a stage name and add parallel and sequential blocks:

The stage is named **Stage1** by default. If you want, you can change the name.

1. Double-click the name.

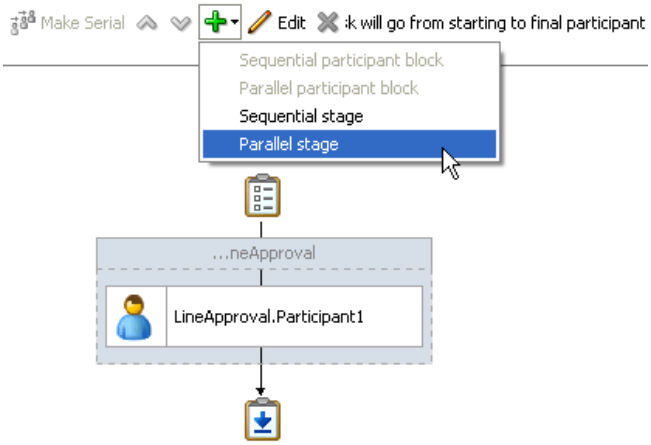
   The Edit dialog shown in Figure 27–21 appears.

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.

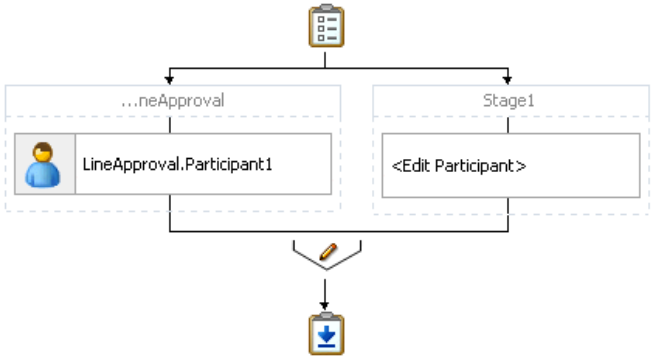
Figure 27–22 Add a Second Stage



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.

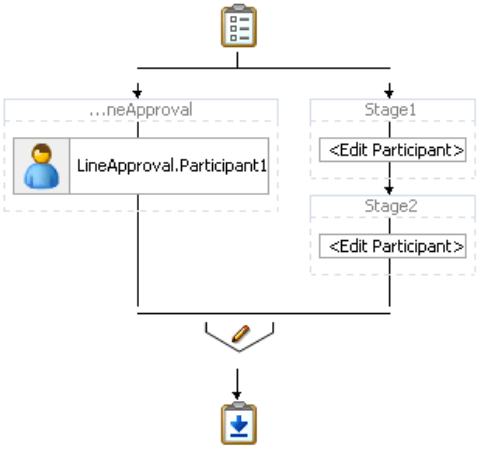
Figure 27–23 Parallel Stage



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:

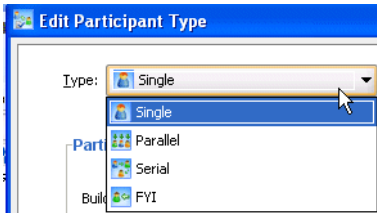
1. In the **Assignment** section, perform one of the following tasks:
   a. Highlight the block below the stage box and click the **Edit** icon. The first time you create a task participant, the box is labeled **<Edit Participant>**.

      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.

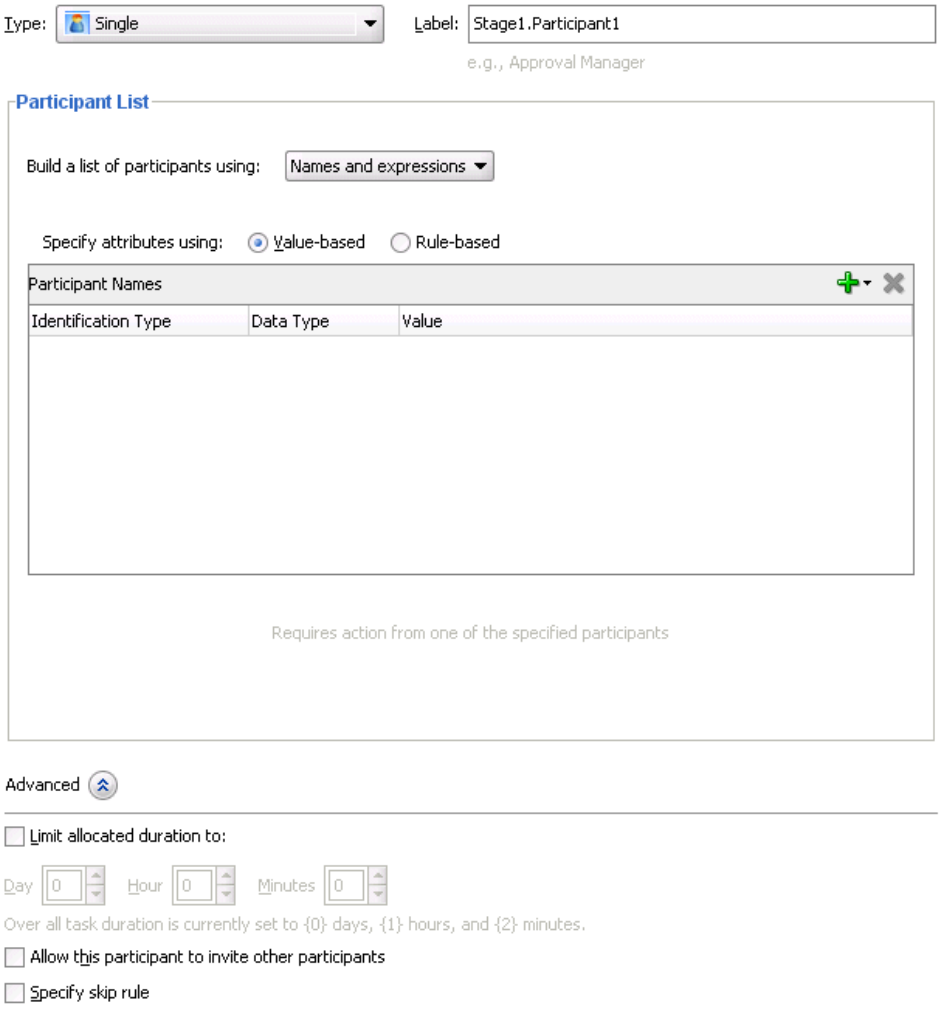
Table 27–6 Participant Types

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



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.

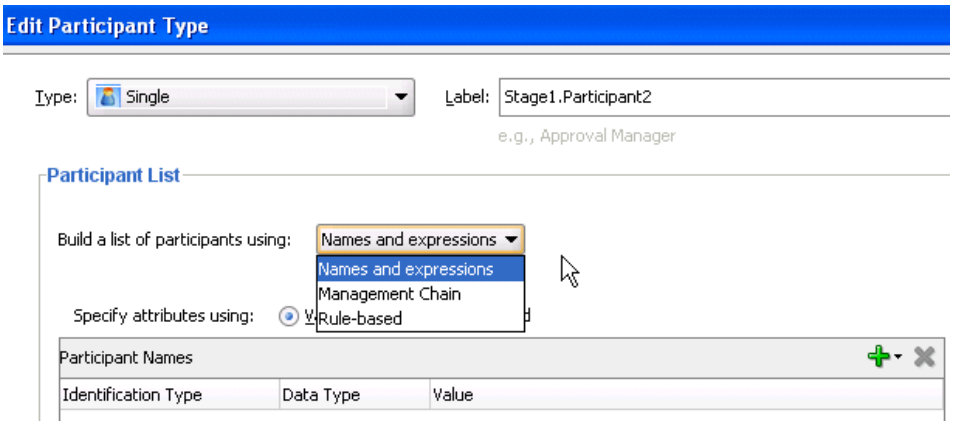
- Rule-based names and expression lists and management chain lists

  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

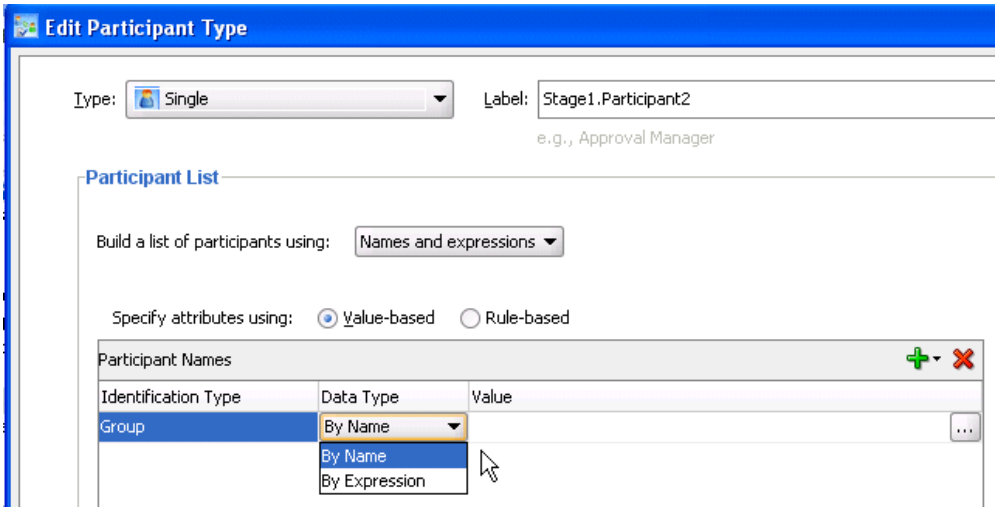
roles), as shown in Figure 27–27. The three selections described above are available: **Names and expressions**, **Management Chain**, and **Rule-based**.

Figure 27–27 Build a List of Participants



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



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:

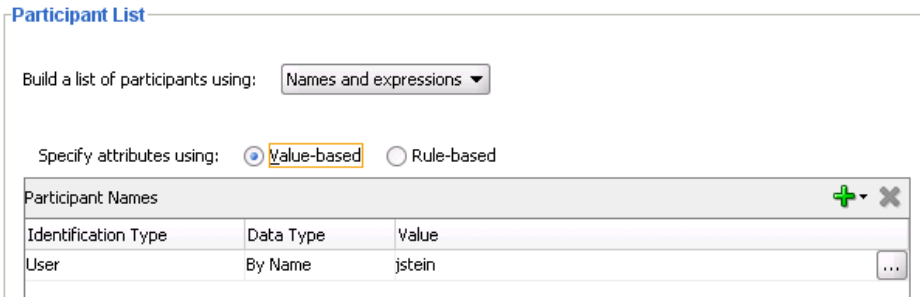
- Users, groups, or application roles, see Section 26.2.1.1.3, "Participant Assignment."
- Statically and dynamically assigning task participants, see Section 26.2.1.2, "Static, Dynamic, and Rule-Based Task Assignment."

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



3. Click the **Add** icon and select a user, group, or application role as a task participant.

   The **Identification Type** column of the **Participant Names** table displays your selection of user, group, or application role.
4. 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:

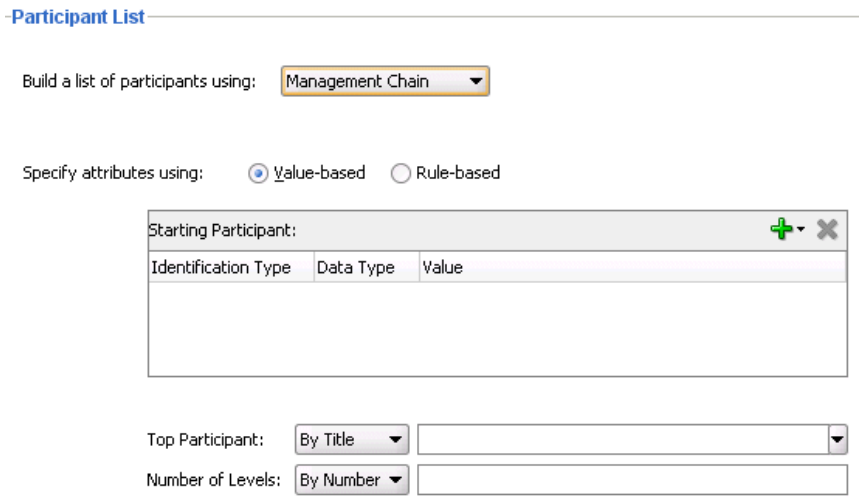
- Users, groups, or application roles, see Section 26.2.1.1.3, "Participant Assignment."
- Statically and dynamically assigning task participants, see Section 26.2.1.2, "Static, Dynamic, and Rule-Based Task Assignment."
- Management chains, see Section 27.3.6.1.1, "Creating a Single Task Participant 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



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

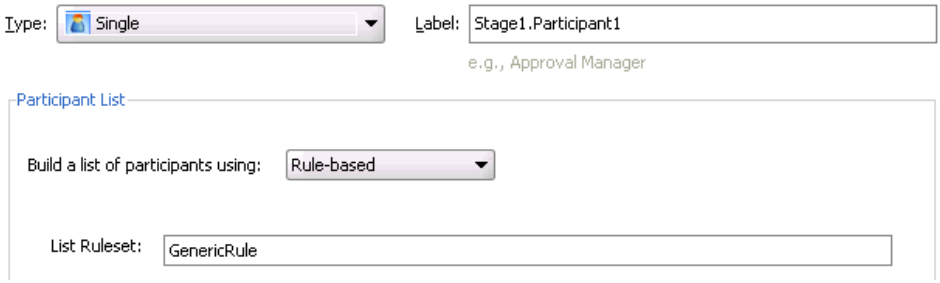
  4. Click **OK**.

- 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



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:

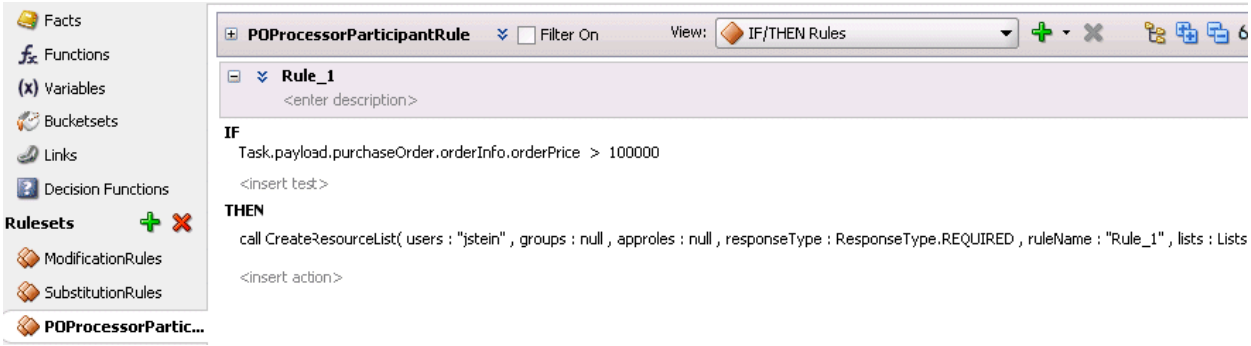
- `CreateResourceList`
- `CreateManagementChainList`

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.

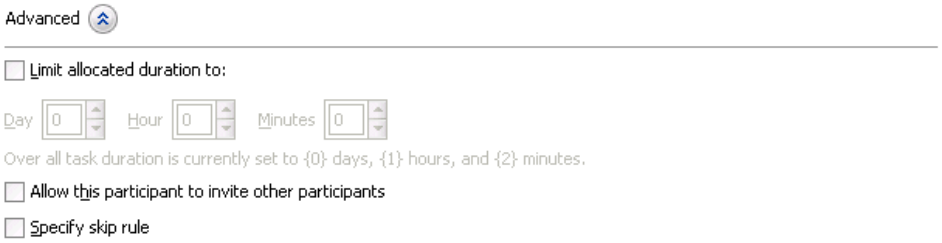
27.3.6.1.2 Specifying a Time Limit for Acting on a Task

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)

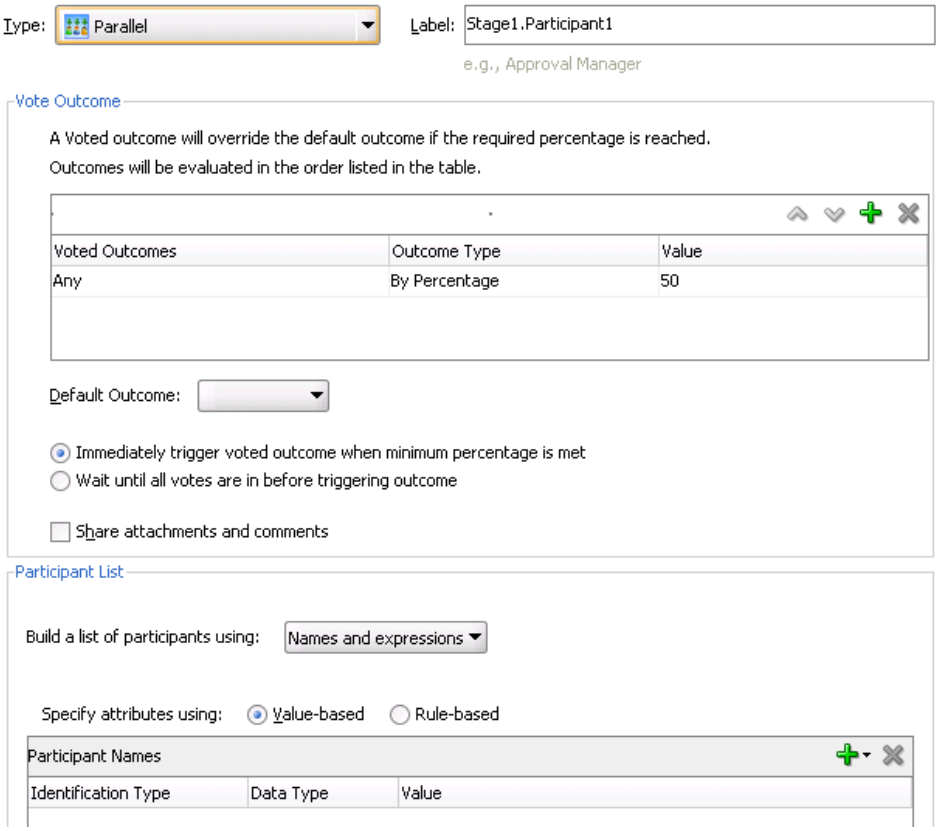
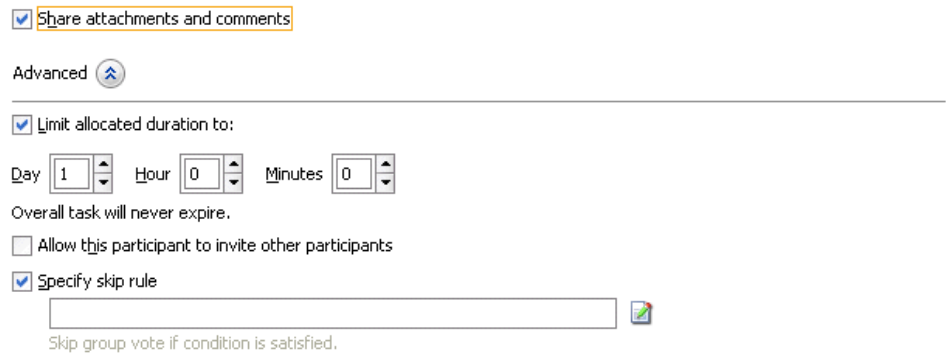


Figure 27–37 Edit Participant Type — Parallel Type (Lower Section of Dialog)



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.

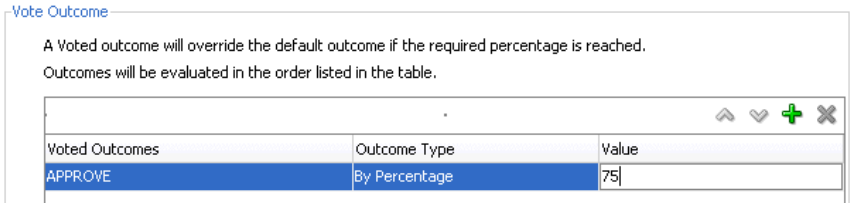
To specify group voting details:

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



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.

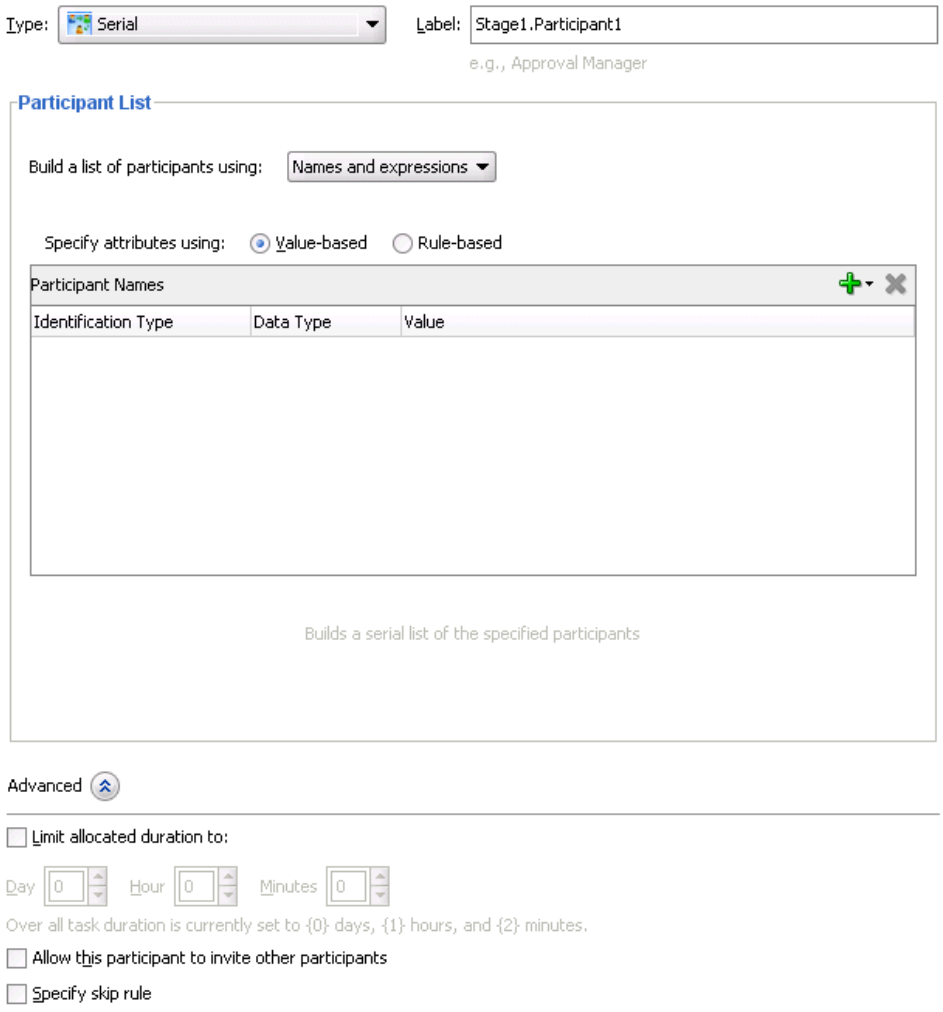
   For information about a valid XPath expression for skipping a participant, see Section 27.3.6.1.4, "Bypassing a Task Participant."

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



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"

27.3.6.3.1 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."

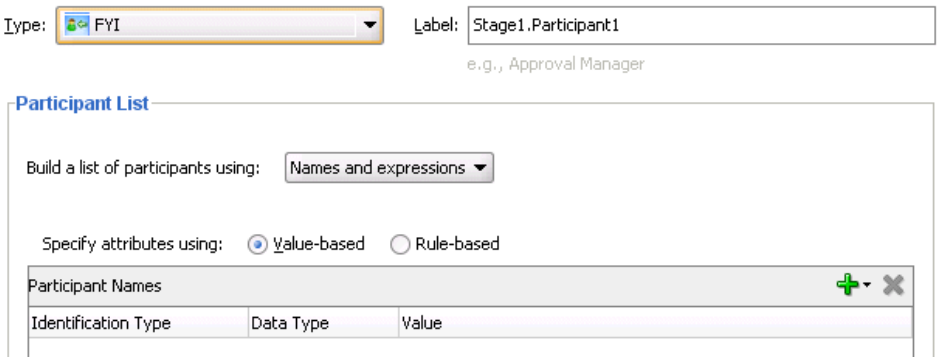
27.3.6.4 Configuring the FYI Participant Type

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



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:

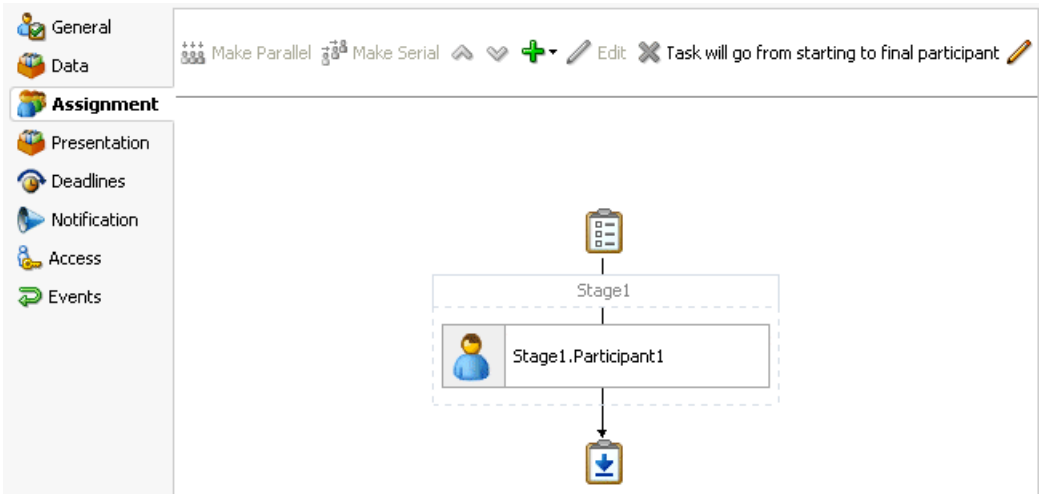
- Value-based name and expression lists
- Value-based management chain lists
- Rule-based names and expression lists
- Rule-based management chain lists
- Rule-based lists

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

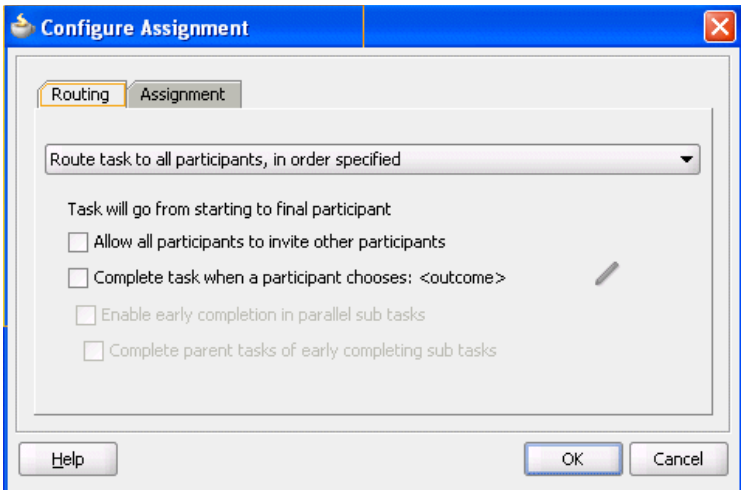


Table 27–10 describes the routing policy methods provided.

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"

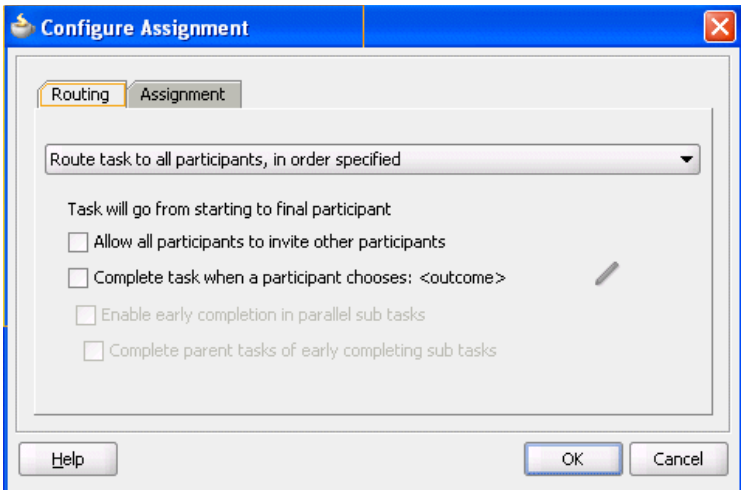
27.3.7.1 Routing Tasks to All Participants in the Specified Order

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.

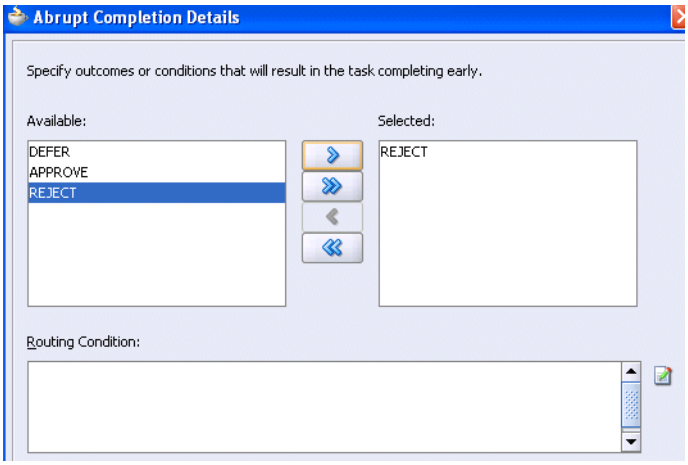
   The Abrupt Completion Details dialog appears.

   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.

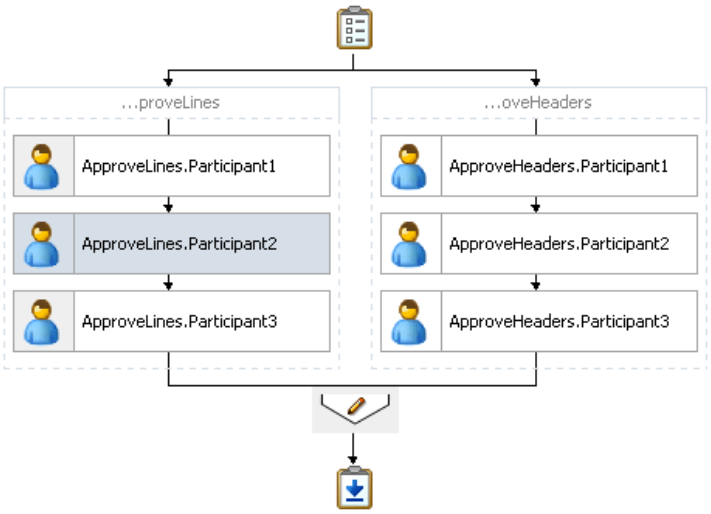
27.3.7.1.3 Enabling Early Completion in Parallel Subtasks

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: ▪ `actualParticipant`: The name of the participant who set the task outcome (for example, `jstein`) ▪ `logicalParticipant`: The logical name (or label) for the routing slip participant responsible for setting the task outcome (for example, `assignee1`) ▪ `outcome`: The outcome that was set (for example, approve or reject) ▪ `level`: If the previous participant was part of a management chain, then this attribute records their level in the chain, where 1 is the first level in the chain. For other participant types, the value is -1. ▪ `totalNumberOfApprovals`: The total number of users that have 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, `Approver1`) to which to route the task.
`COMPLETE`	Finishes routing and completes the task. The task is marked as completed, and no further routing is required.	None
`ESCALATE`	Escalates and reassigns the task according to the task escalation policy (usually to the manager of the current assignee).	None

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:

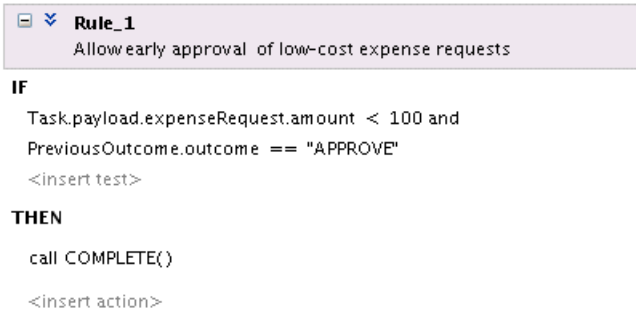
- If the total amount of the expense request is less than $100, approval is only required from one of the participants. Otherwise, it must be approved by all three.
- If an expense request is rejected by any of the participants, it must be returned to the previous participant for re-evaluation. If it 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:

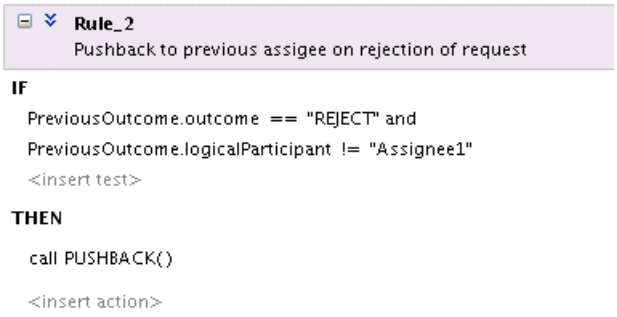
- Early approval rule (Figure 27–46):

Figure 27–46 Early Approval Rule



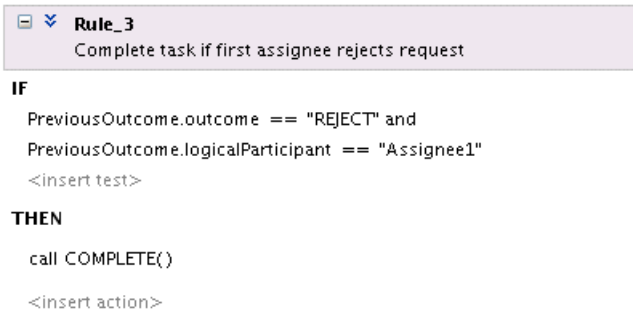
- Push back on the rejected rule (Figure 27–47):

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

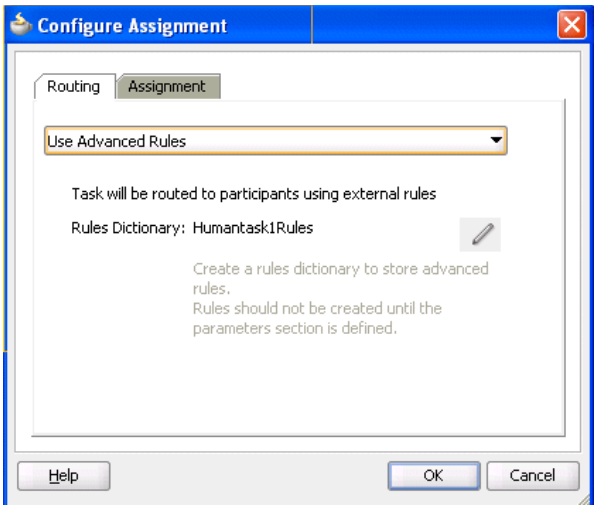
For more information about customizations, see Chapter 43, "Customizing SOA Composite Applications."

27.3.7.2.6 Creating Advanced Routing Rules

To create advanced routing rules:

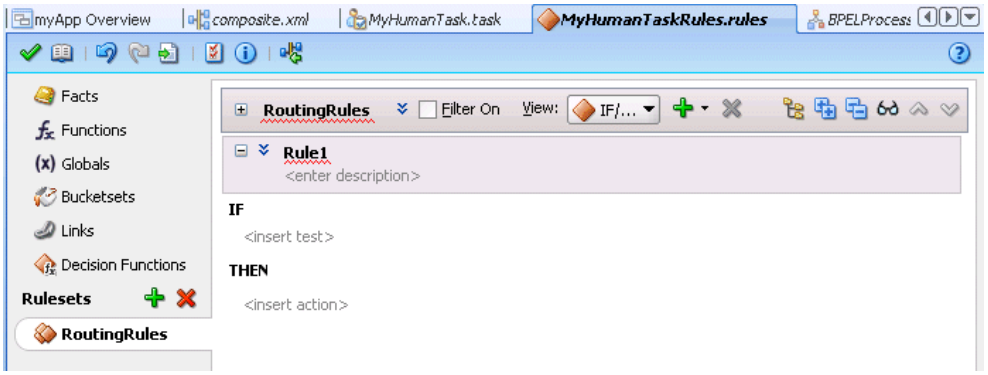
1. In the **Assignment** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Use Advanced Rules** from the list.
3. To the right of **Rules Dictionary**, click the **Edit** icon, as shown in Figure 27–49.

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.

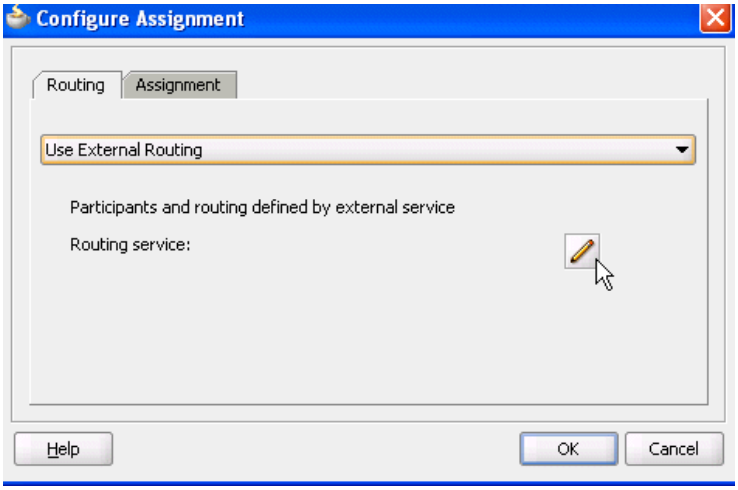
Use this option if you do not want to use any of the routing rules to determine task assignees. In this case, all the logic of task assignment is delegated to the external routing service.

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

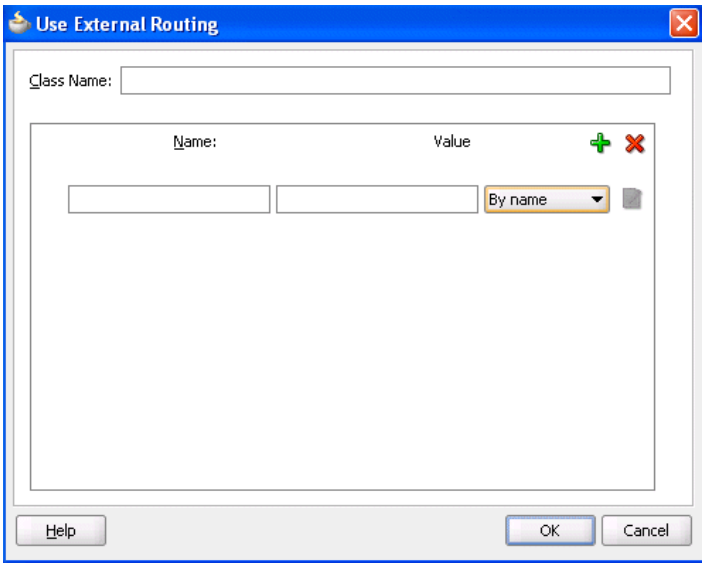
1. In the **Assignment** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Use External Routing** from the list.
3. Click the **Edit** icon, as shown in Figure 27–51.

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.

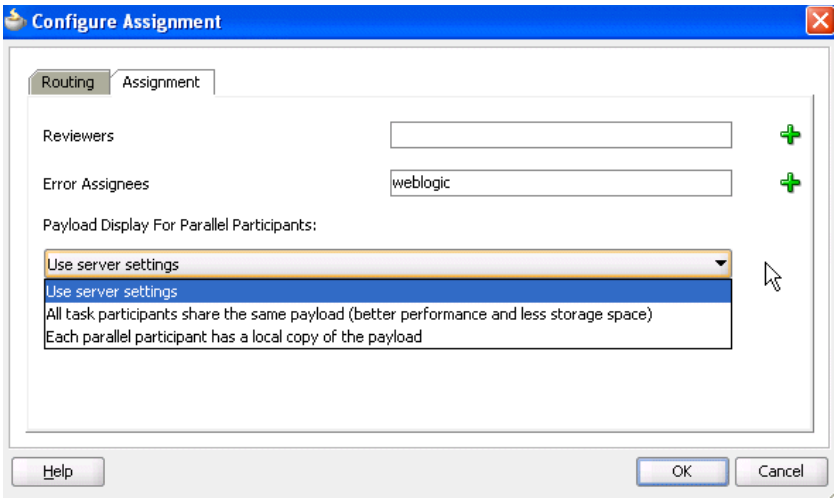
If there are any errors in evaluating the error assignees, the task is marked as being in error.

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:

1. 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.
2. 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:

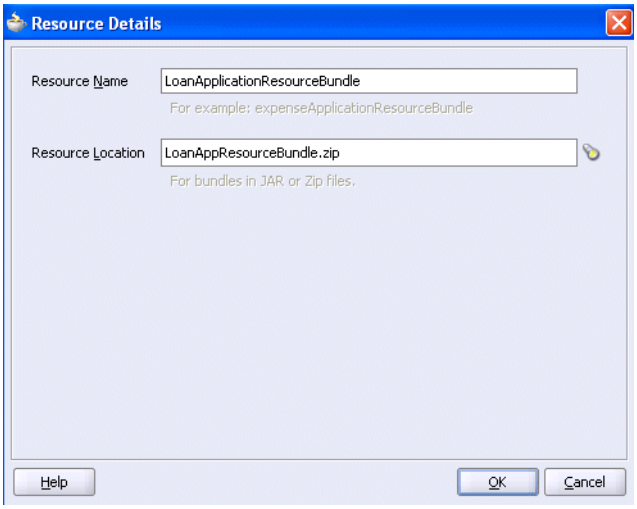
- `APPROVE=Approve`
- `REJECT=Reject`

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

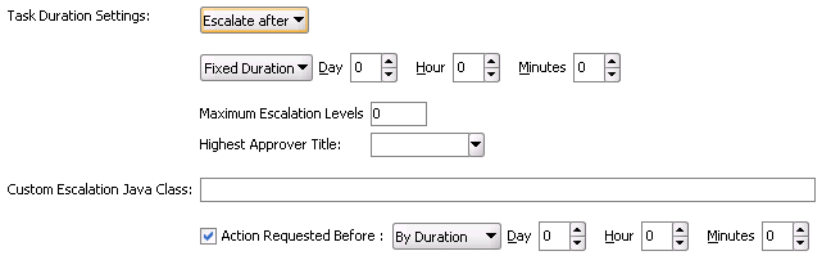
Figure 27–56 shows the **Deadlines** section of the Human Task Editor.

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

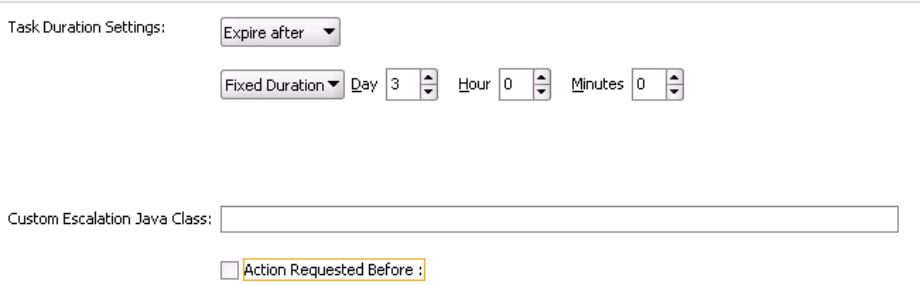


27.3.9.1 Introduction to Escalation and Expiration Policy

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



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:

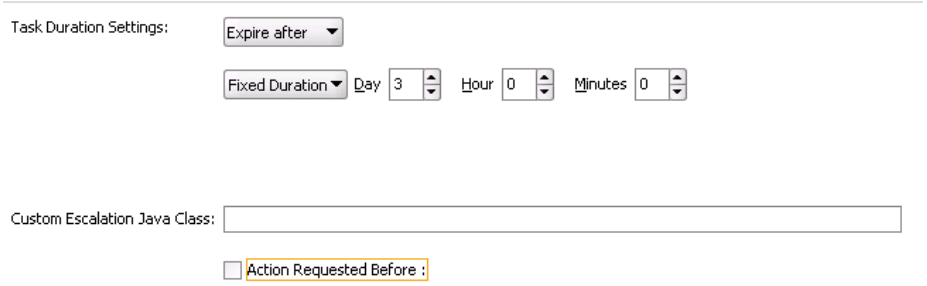
1. In the dropdown list of the **Deadlines** section, select **Expire after**, as shown in Figure 27–58.
2. Specify the maximum time period for the task to remain open.

   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



27.3.9.4 Extending an Expiration Policy Period

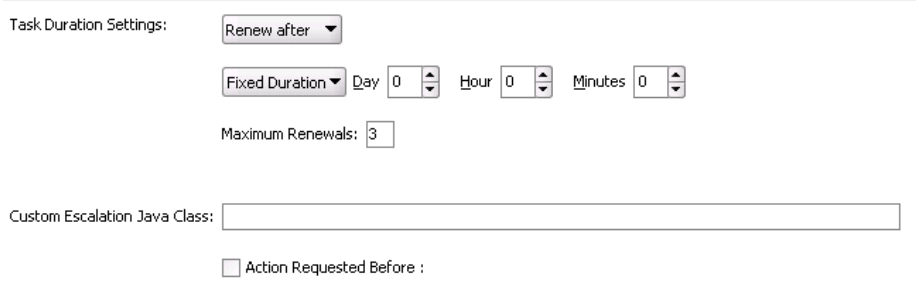
You can extend the expiration period when the user does not respond within the allotted time. You do this by specifying the number of times the task can be renewed upon expiration (for example, renew it an additional three times) and the duration of each renewal (for example, three days for each renewal 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



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**

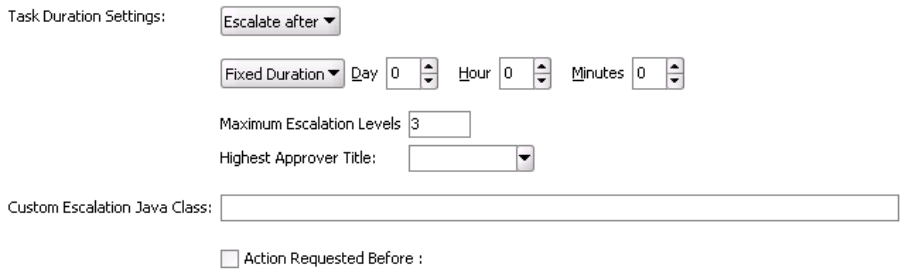
  Number of management levels to which to escalate the task. This field is required.

- **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



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."

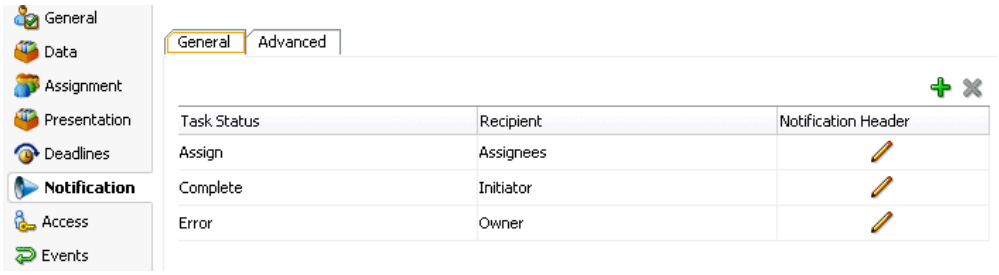
27.3.10 How to Specify Participant Notification Preferences

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.

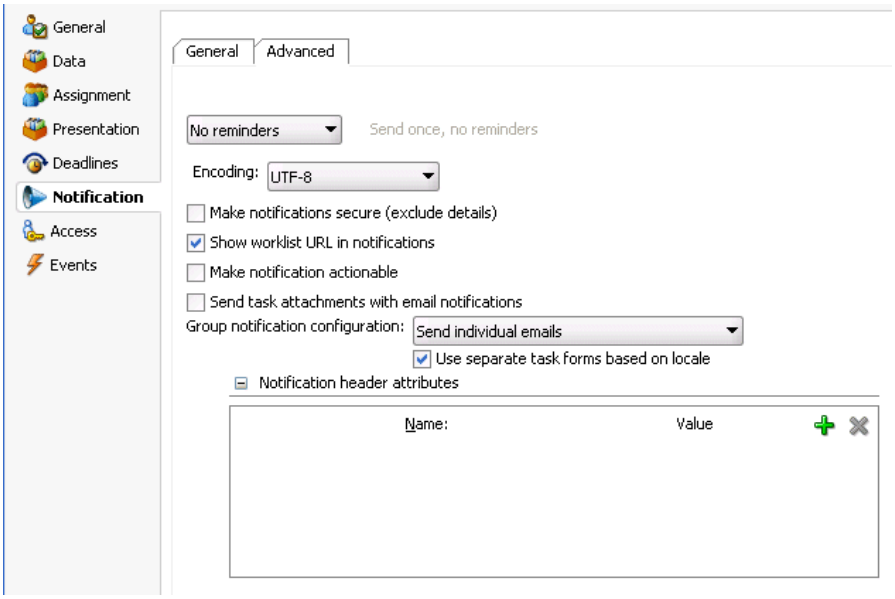
Table 27–15 Human Task Editor — General Tab of Notification Section

For This Subsection...	See...
Task Status **Recipient**	Section 27.3.10.1, "Notifying Recipients of Changes to Task Status"
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

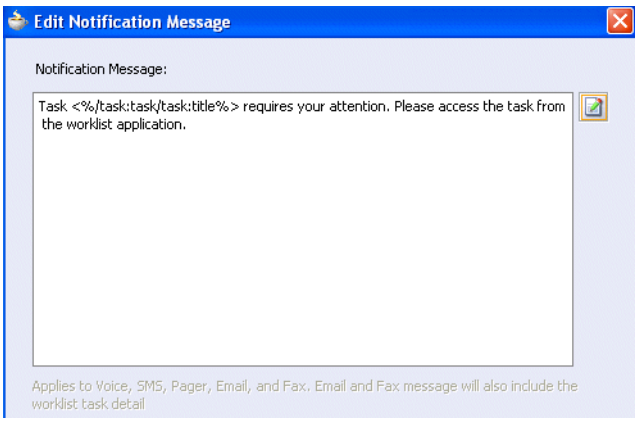
A default notification message is available for delivery to the selected recipient. If you want, you can modify the default message text.

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;
```

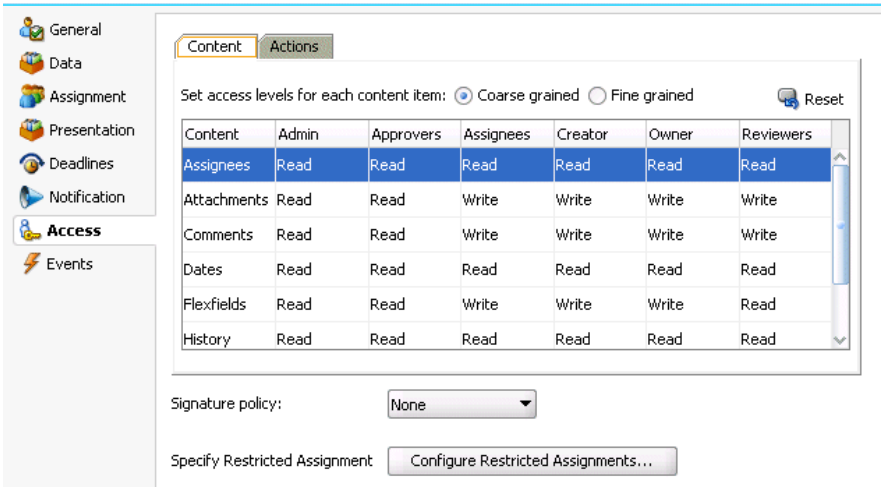
For more information about this method, see *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle SOA Suite*.

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



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**).

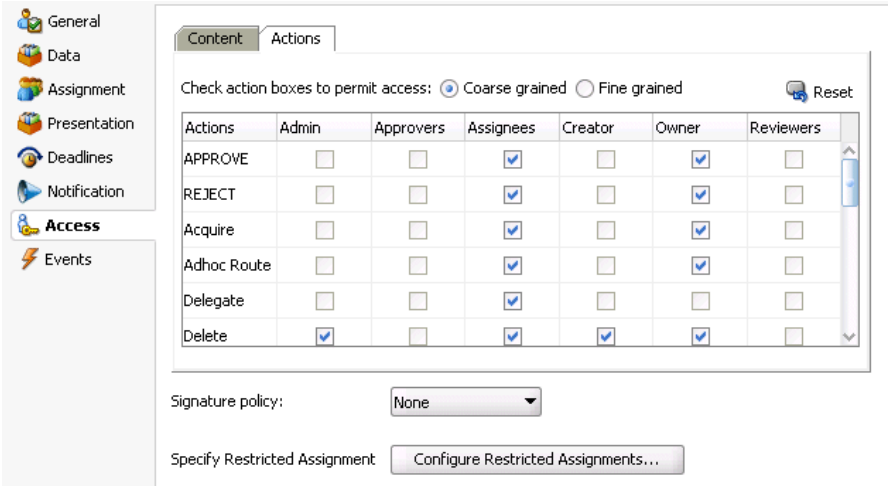
Note: Access rules are always applied on top of what the system permits, depending on who is performing the action and the current state of the task.

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



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

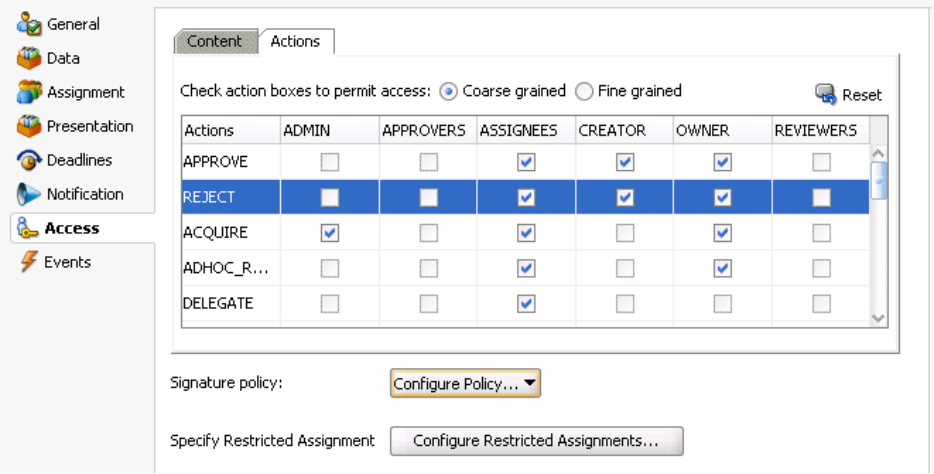
Digital signatures provide a mechanism for the nonrepudiation of digitally-signed human tasks. This ability to mandate that a participant acting on a task signs the details and their action before the task is updated ensures that they cannot repudiate it later.

> **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



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"

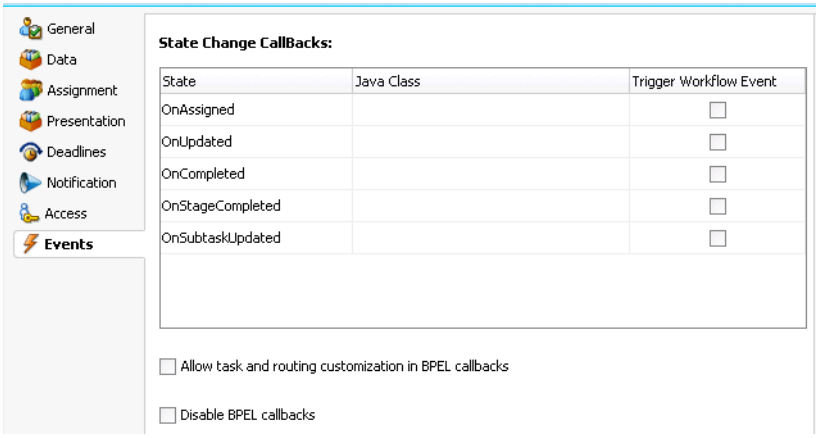
27.3.14.1.1 Specifying Java Callbacks

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



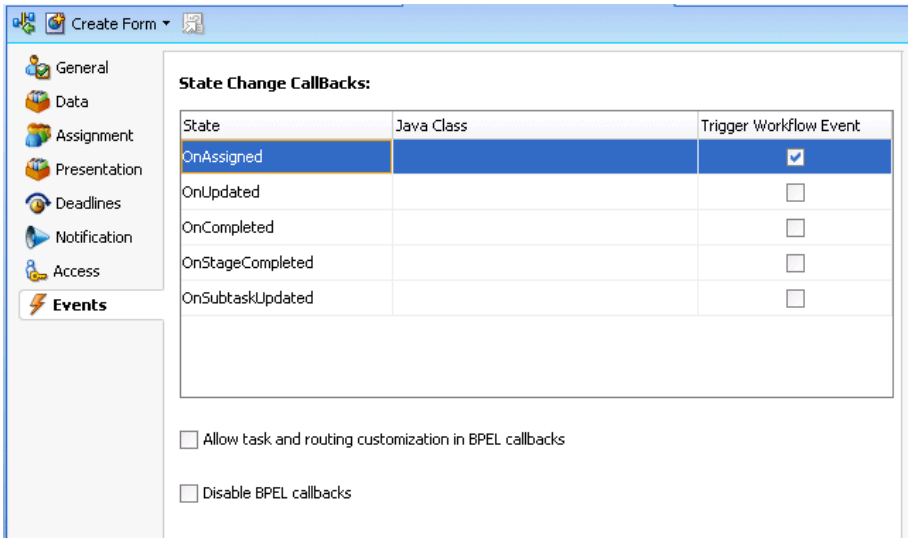
3. Click **OK**.

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



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`.

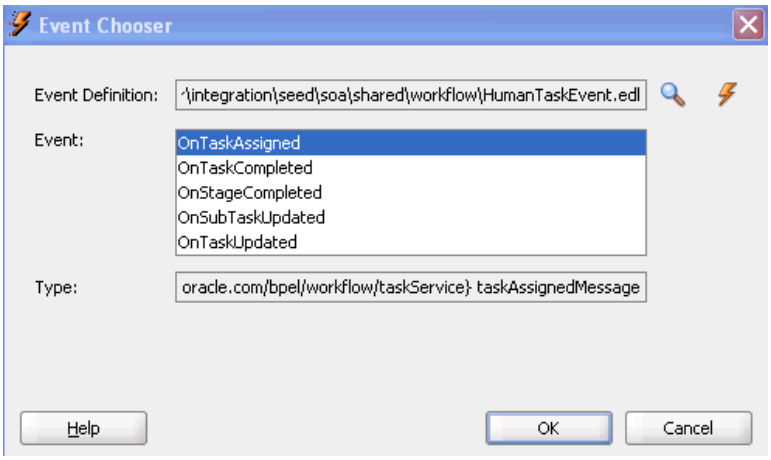
---

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**.

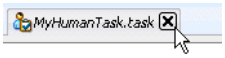
27.3.17 How to Exit the Human Task Editor and Save Your Changes

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:

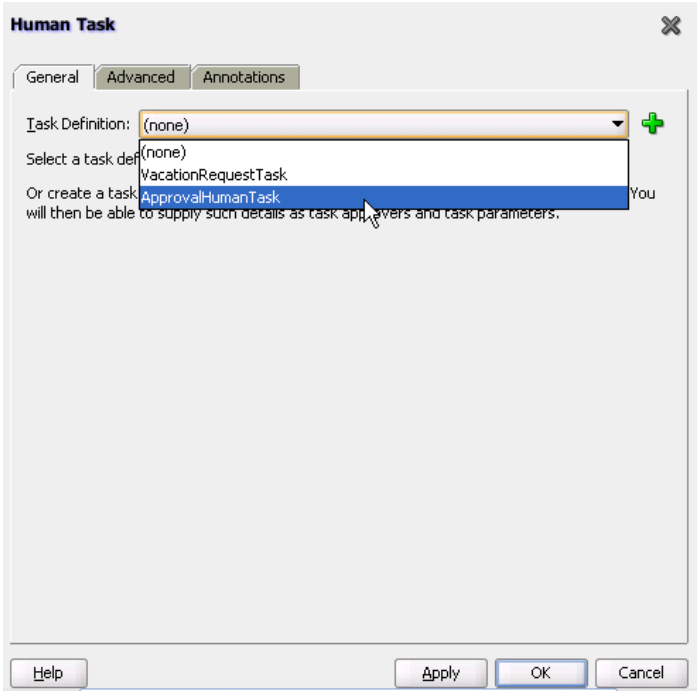
1. Go to the SOA Composite Editor.
2. Double-click the BPEL process service component with which to associate the `.task` file of the human task service component.
3. In the Component Palette, expand **SOA Components**.
4. Drag a new **Human Task** activity into the BPEL process.

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.

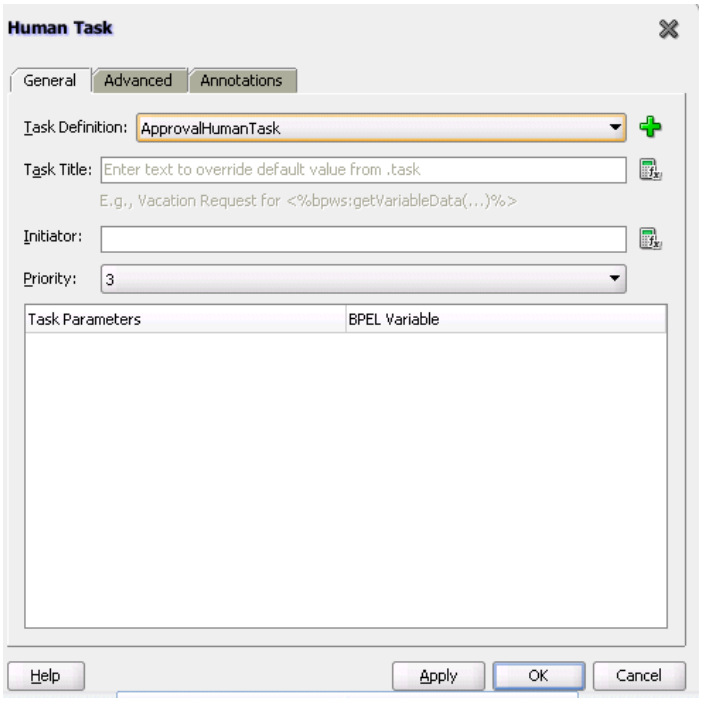
If you then drag and drop a human task service component into the BPEL process to use the same **taskSwitch** switch activity, a new **taskSwitch** switch activity is created.

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 **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:

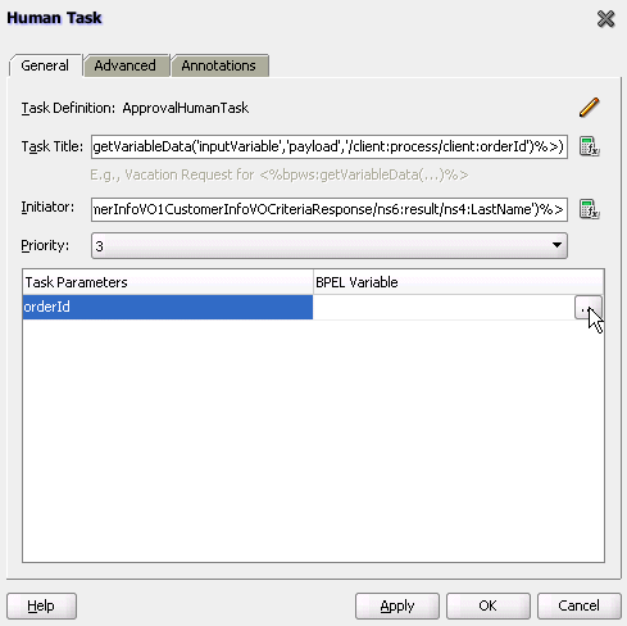
1. To the right of the **Initiator** field of the **General** tab, enter the initiator (for example, `jcooper`) or click the icon to display the Expression Builder dialog for dynamically specifying an initiator. This field is optional. If not specified, the initiator defaults to the task owner specified on the **Advanced** tab of the Human Task dialog. The initiator defaults to `bpeladmin` if a task owner is also not specified.
2. From the **Priority** list, select a priority value between **1** (the highest) and **5**. This field is provided for user reference and does not make this task a higher priority during runtime. 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



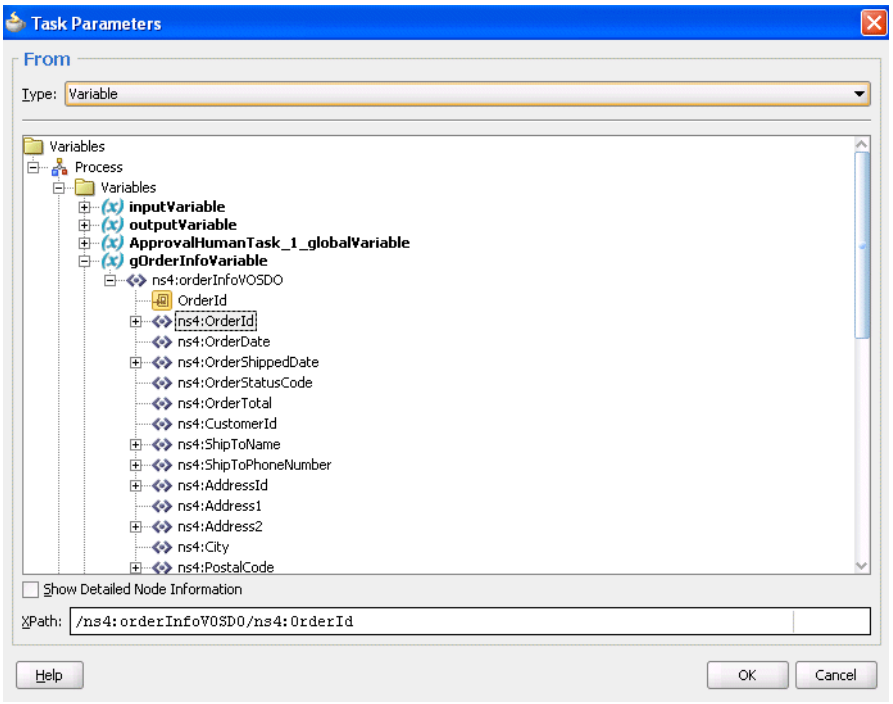
To specify task parameters:

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.

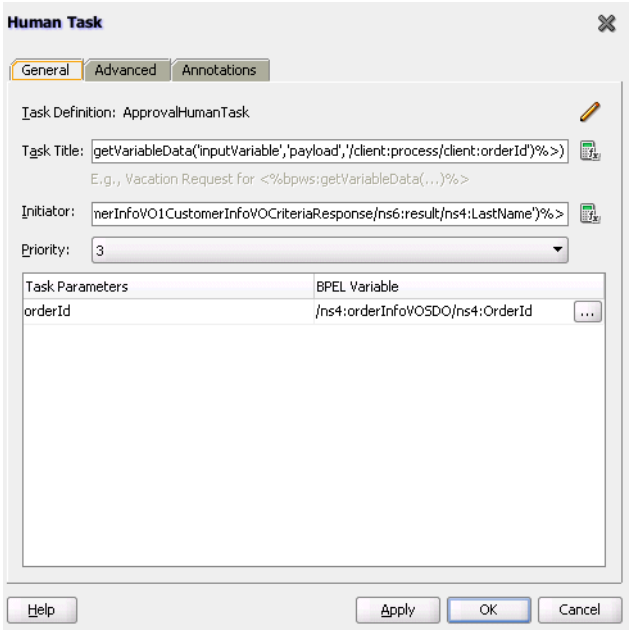
Figure 27–74 Variables Tree



3. Click **OK**.

   The Human Task dialog shown in Figure 27–75 appears as follows.

Figure 27–75 Human Task Dialog

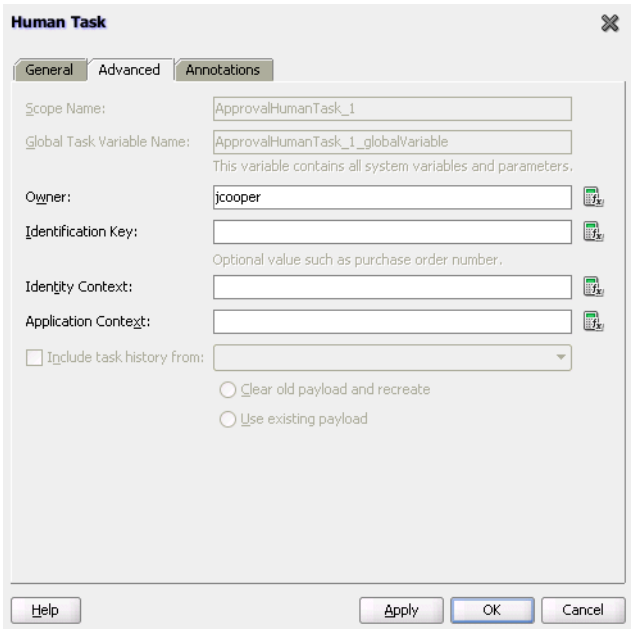


4. To define advanced features for the human task activity, click the **Advanced** tab and go to Section 27.4.4, "How to Define the Human Task Activity Advanced Features." Otherwise, click **OK** to close the Human Task dialog.

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 **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 **Global Task Variable Name**	Section 27.4.4.1, "Specifying a Scope Name and a 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**

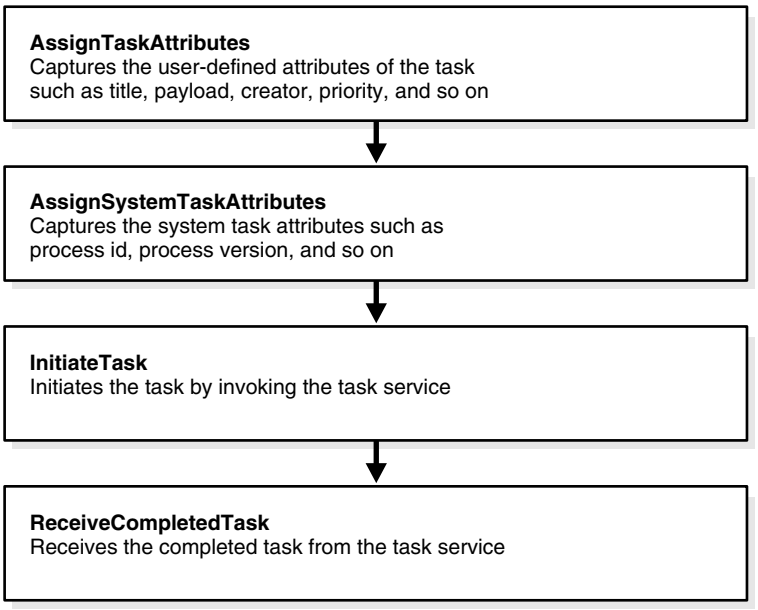
     This option is applicable when the payload attributes in the XML files of the human tasks involved in this extended workflow are the same.

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`

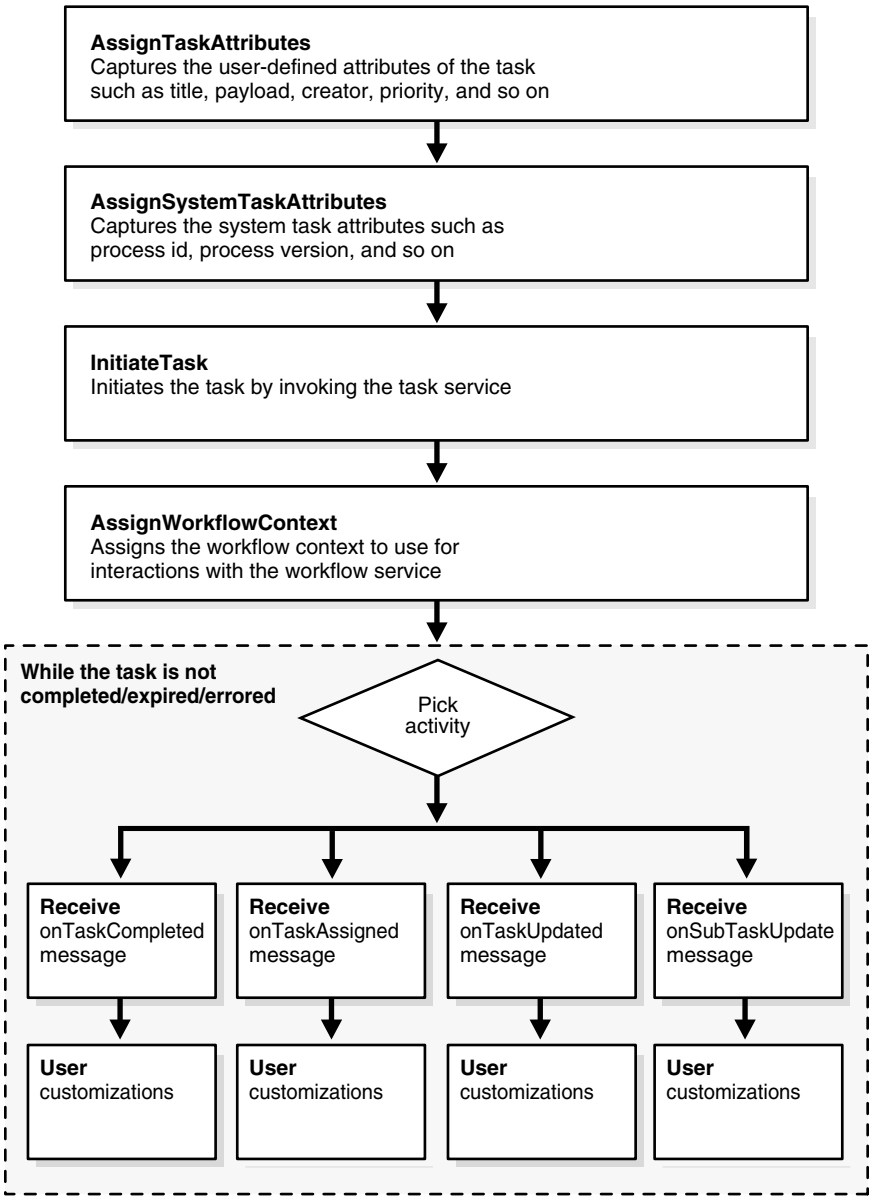
  This callback is invoked for any update to a subtask.

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>
```

28

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:

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*.

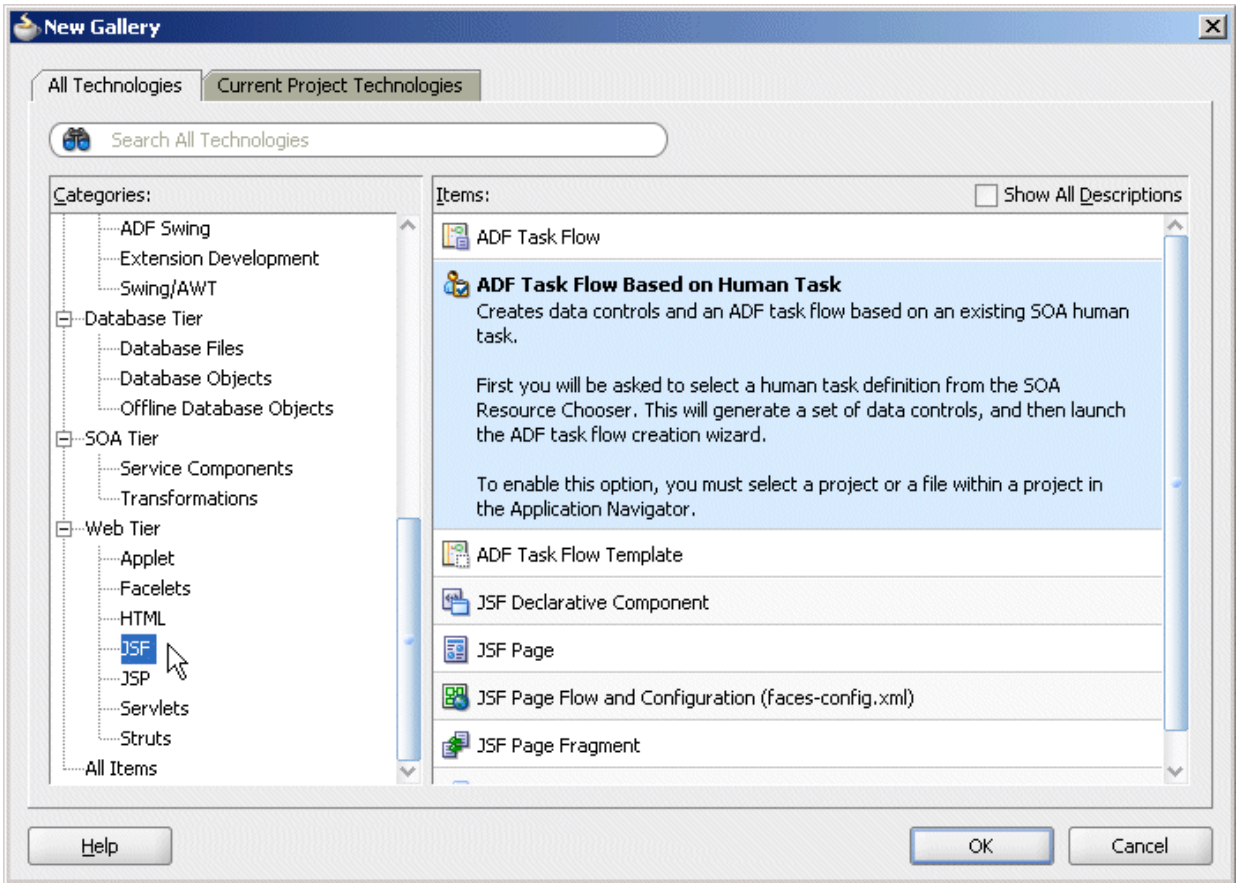
28.1 Introduction to the Task Form

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



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>
```

28.2 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

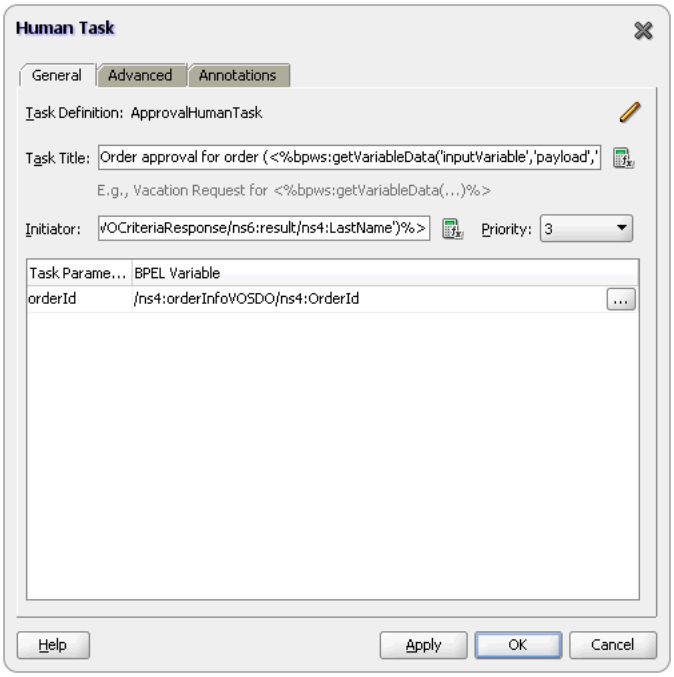
The `.task` file that specifies the human task is easily associated with the task flow when the two are located in the same application.

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**.

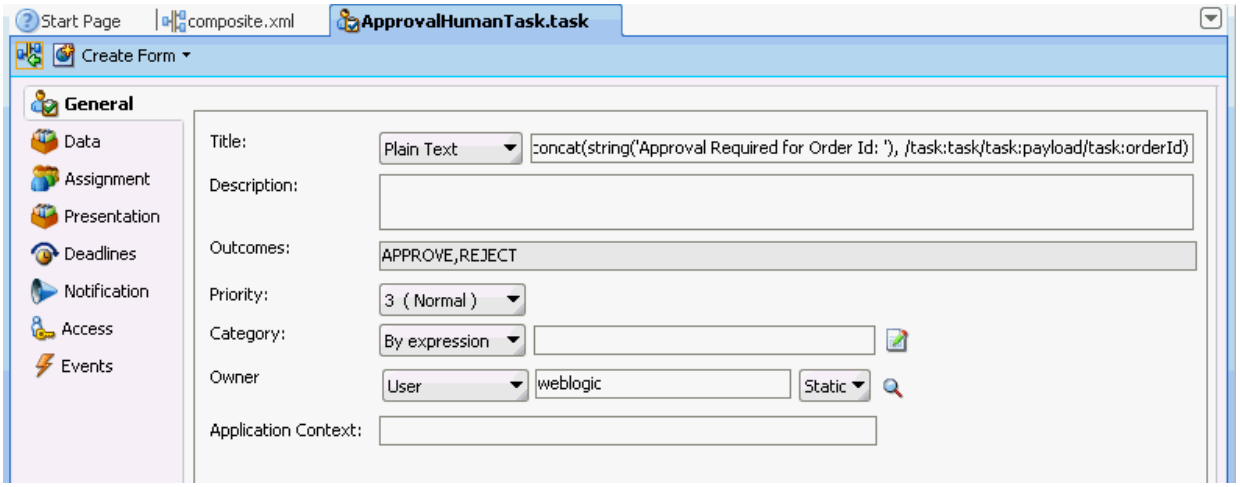
   Figure 28–2 shows the Human Task dialog.

Figure 28–2 Editing a Human Task



3. In the **.task** tab (shown in Figure 28–3), click **Create Form** and select **Auto-Generate Task Form**.

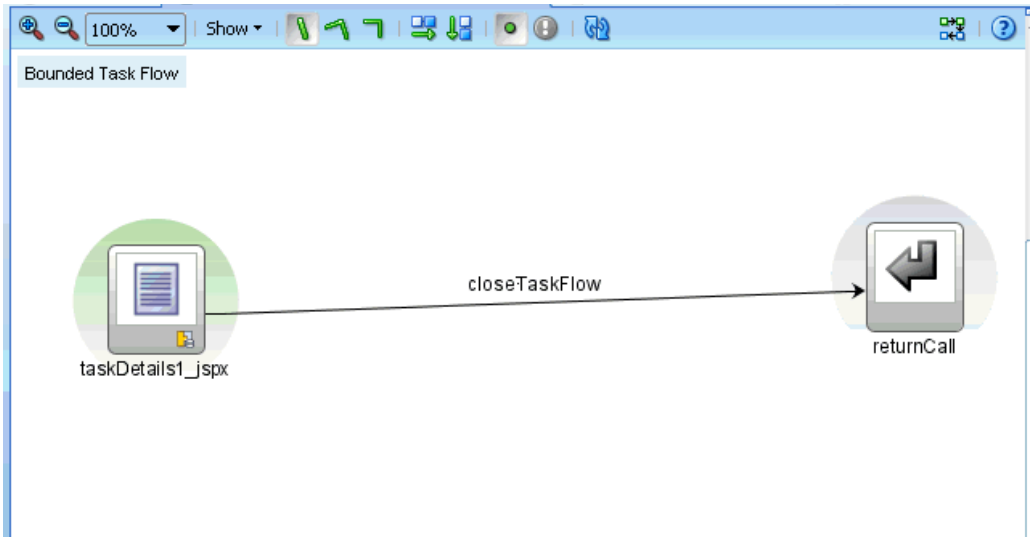
Figure 28–3 Creating a Task Flow from the Human Task Editor



4. Provide a project name and a directory path (or use the default) and click **OK**.

   The **taskDetails1_jspx** icon appears in the designer, as shown in Figure 28–4.

Figure 28–4 The taskDetails1_jspx 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_jspx** icon appears in the designer, as shown in Figure 28–4. 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."

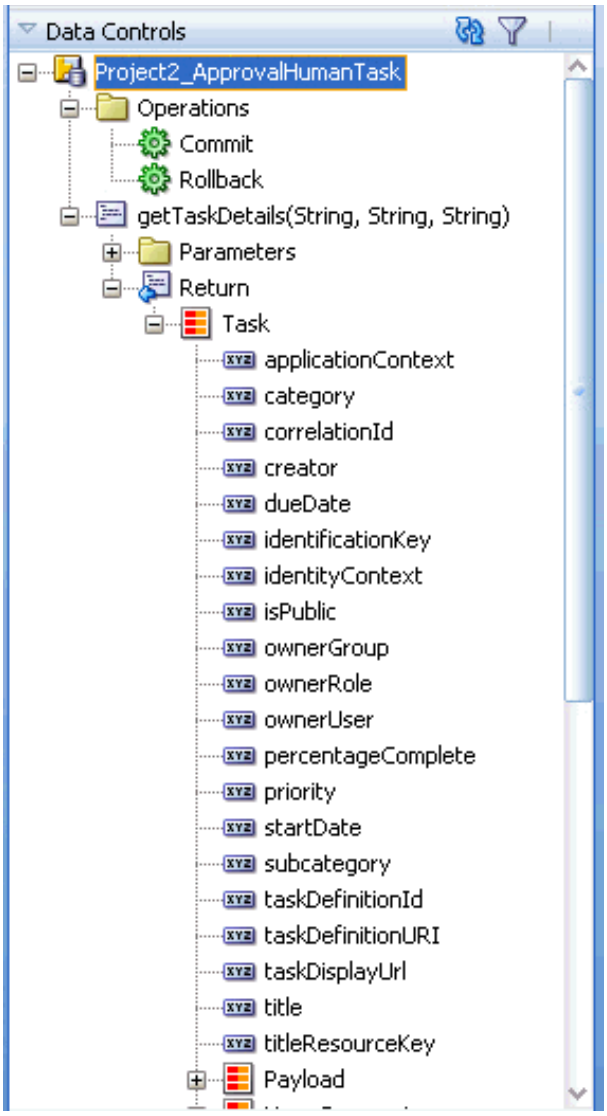
28.3.3 What Happens When You Create an ADF Task Flow Based on a Human Task

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 28–5.

Figure 28–5 The Task Collection in the Data Controls Panel



The data controls for the task (represented by the **Task** node in Figure 28–5) 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"

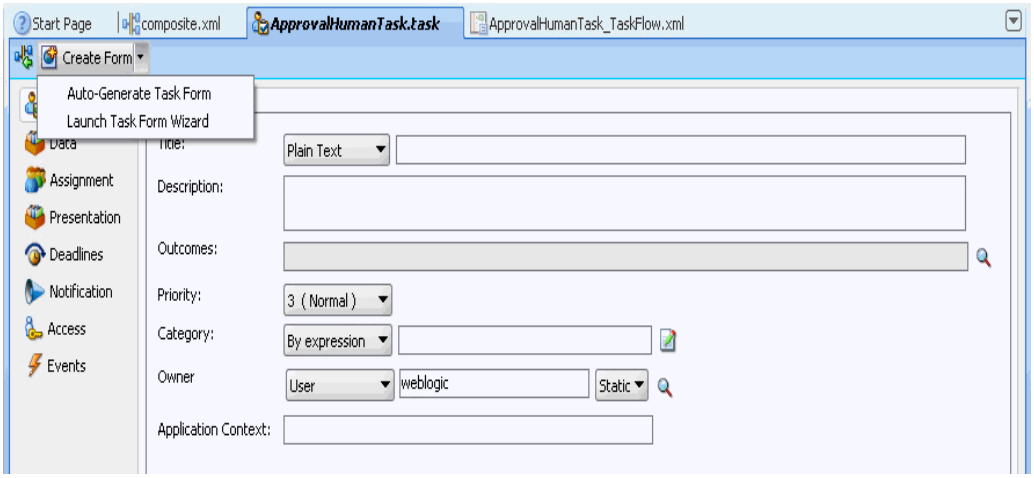
28.4.1 How To Create an Autogenerated Task Form

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 28–6.

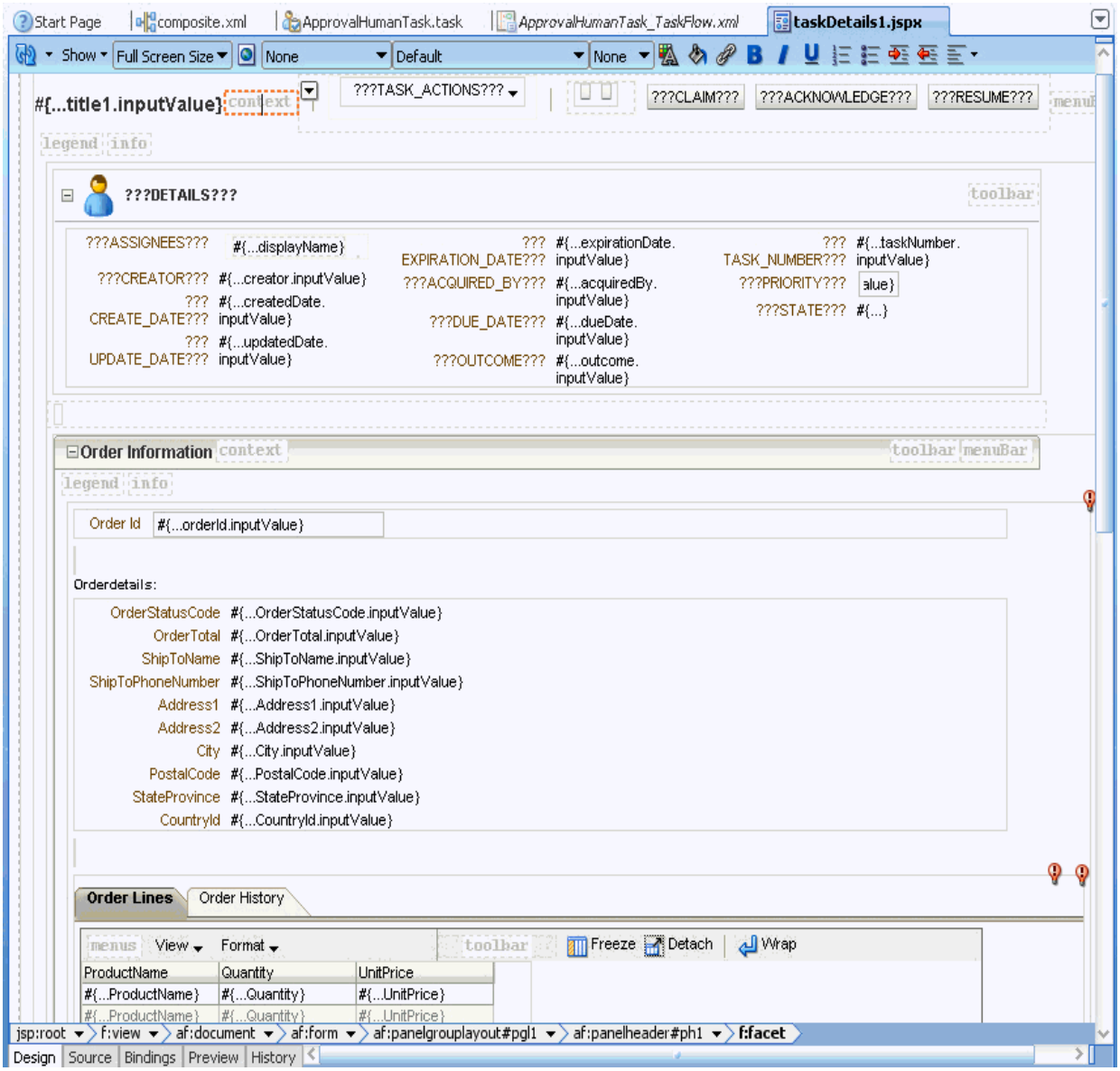
Figure 28–6 Creating a Task Form



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 28–7.

Figure 28–7 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.

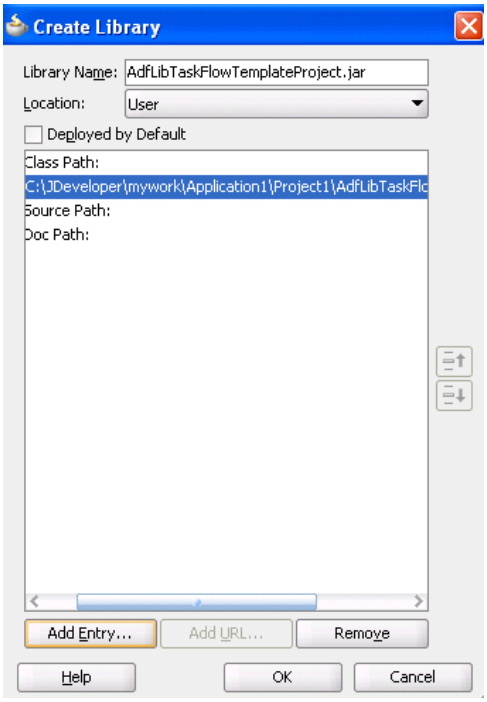
3. Highlight **Class Path**, and click **Add Entry**.

   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 28–8 provides details.

Figure 28–8 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*.

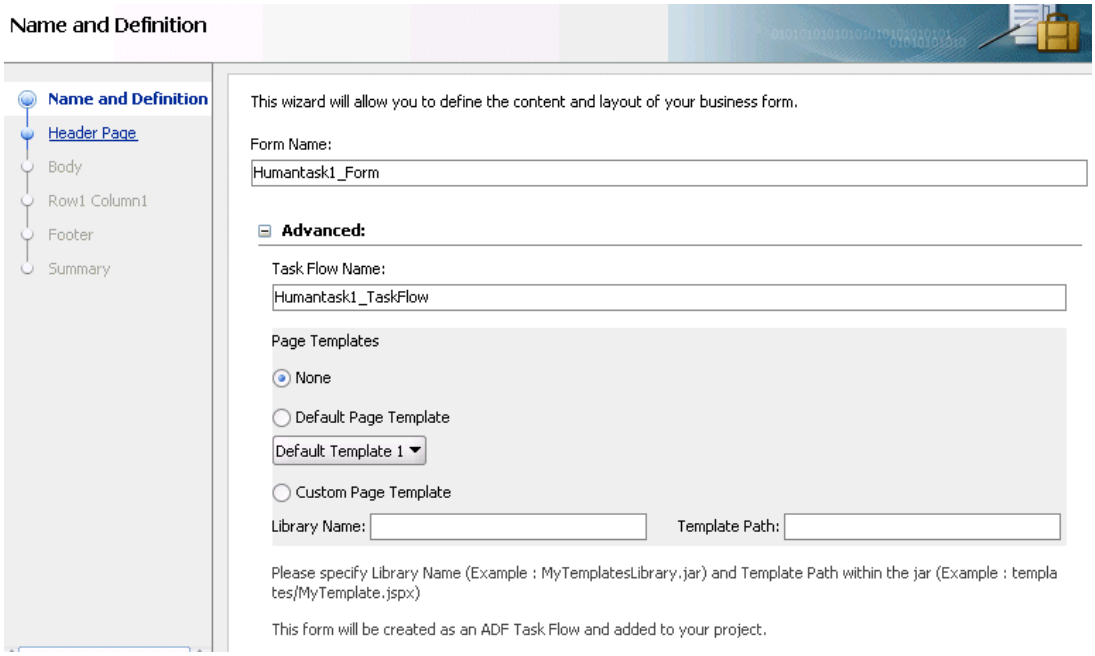
To create a custom task form:

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 28–9, provide the name of the form (`.jspx` file) that is generated at the end of the wizard. The default name, `Humantask`*`number`*`_Form`, is provided if you do not provide a name. Ensure that valid characters are used in the name. Spaces are not permitted.

Figure 28–9 Custom Task Form Wizard: Form Name and Definition

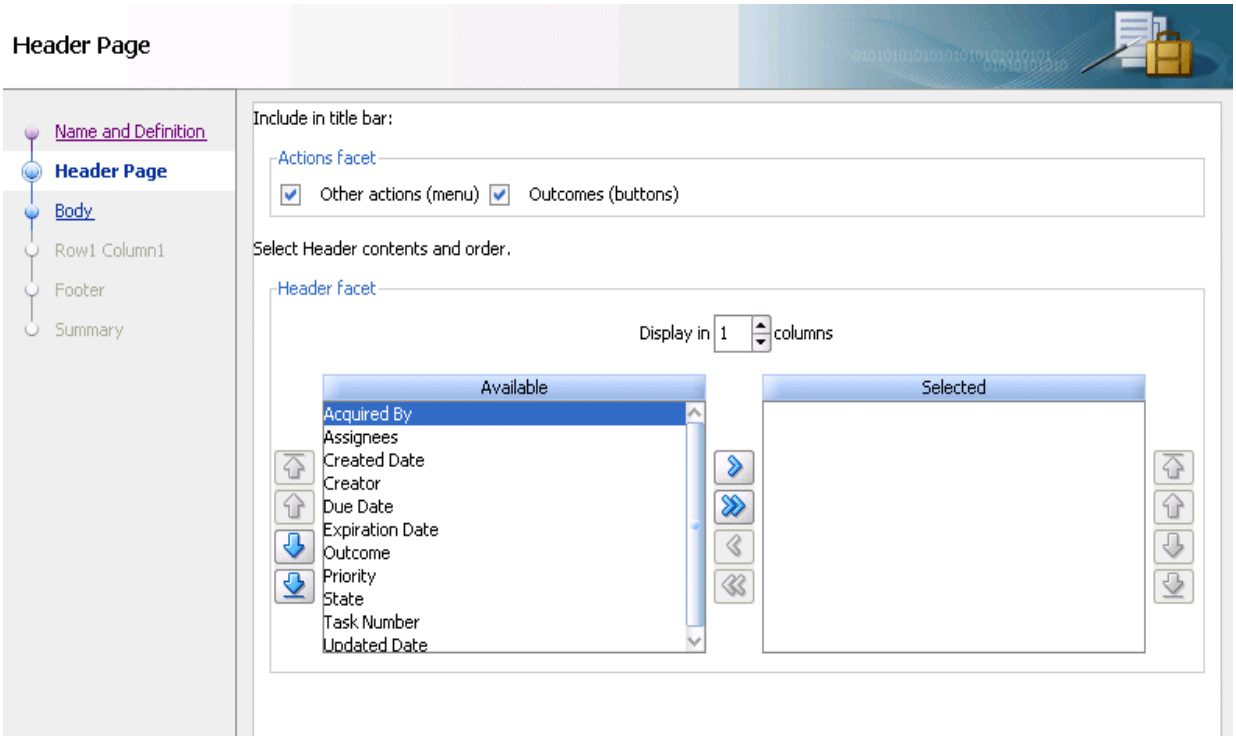


6. If you want to specify advanced definitions, expand **Advanced**.
7. Specify the following information, and click **Next**. This form is created as an ADF task flow and added to the project.
   - **Task Flow Name**: The name of the ADF task flow that is generated at the end of the wizard. Accept the default name of `Humantask`*`number`*`_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.jspx`) 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 28–10, perform the following procedures and click **Next**.

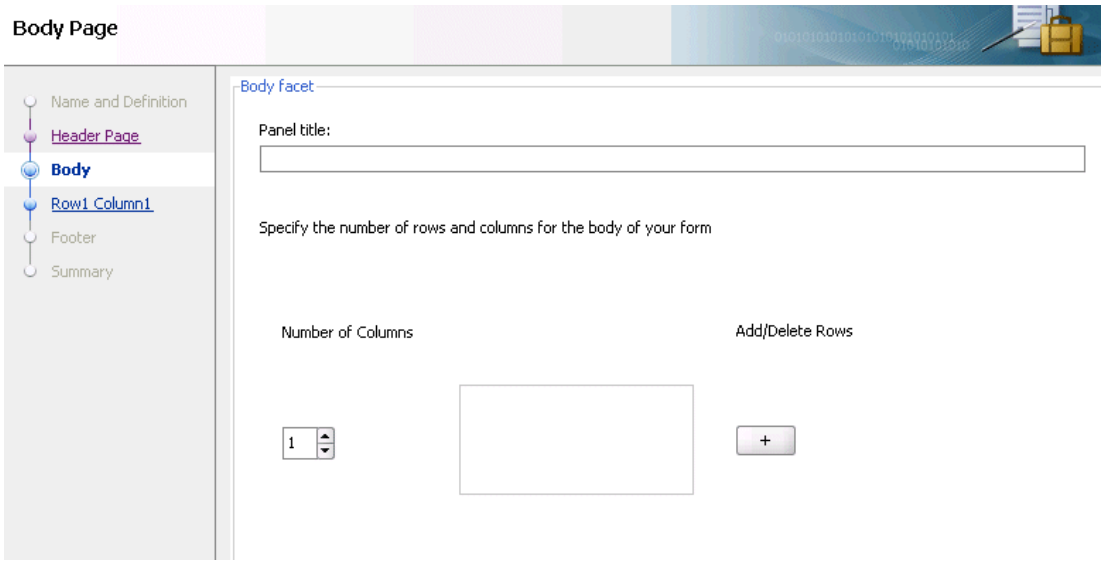
- In the **Actions facet** section, select the options to include in the title bar of the task form:
  - **Other actions (menu)**: Lists the system actions that are possible for the task, such as **Request Information**, **Reassign**, **Renew**, **Suspend**, **Escalate**, and **Save**.
  - **Outcomes (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 28–10 Custom Task Form Wizard: Setting Up the Header



9. On the Body page, shown in Figure 28–11, 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.

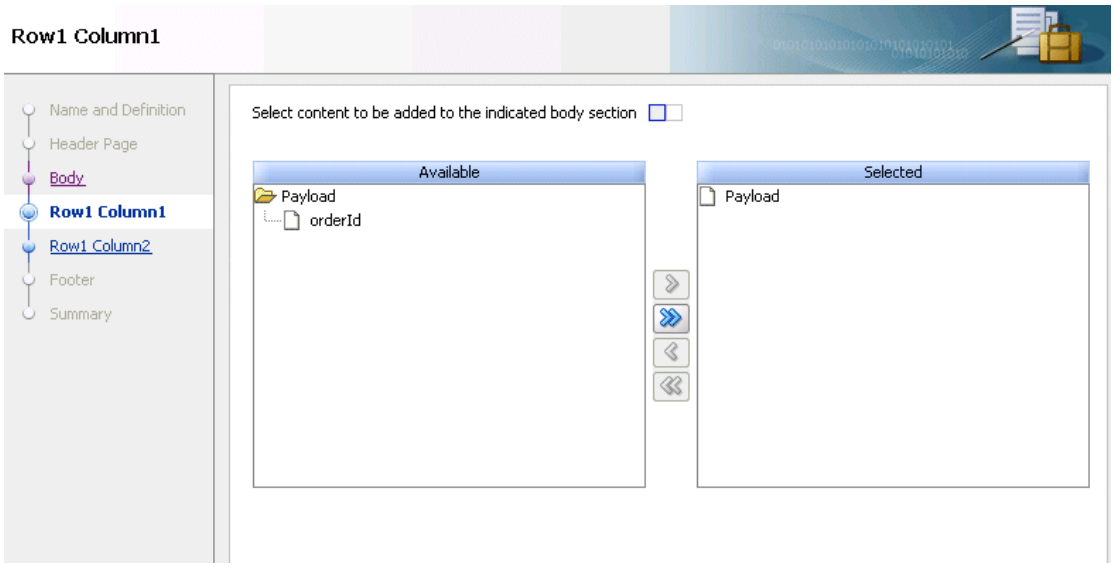
Figure 28–11 Custom Task Form Wizard: Setting Up the Body



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 28–12, move all or part of the payload to the **Selected** list and click **Next**.

Figure 28–12 Custom Task Form Wizard: Selecting the Body Fields

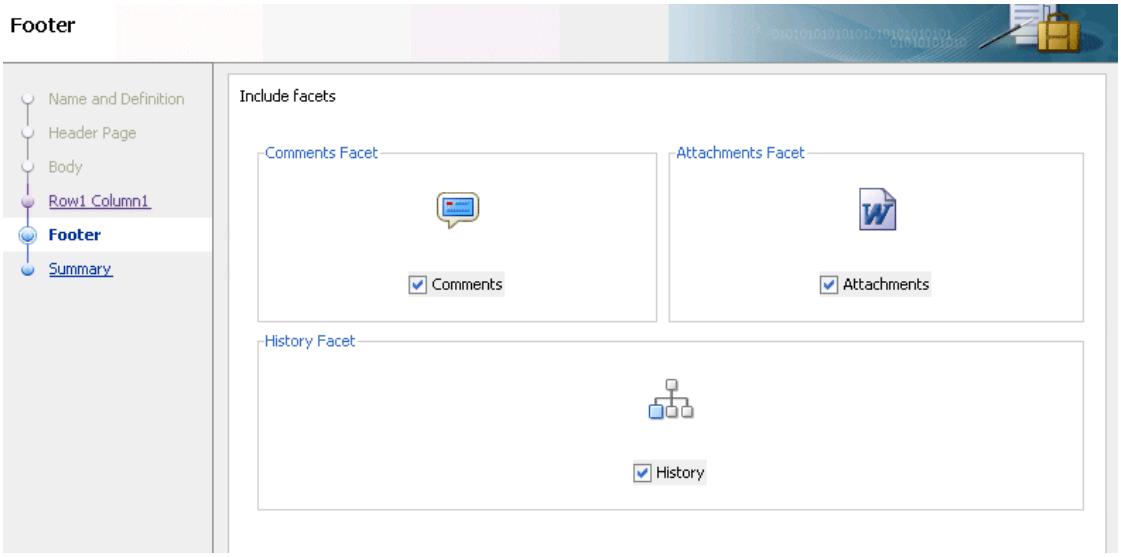


11. For any Row*n* Column*n* page after Row1 Column1, repeat Step 10 and click **Next**.

    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**).

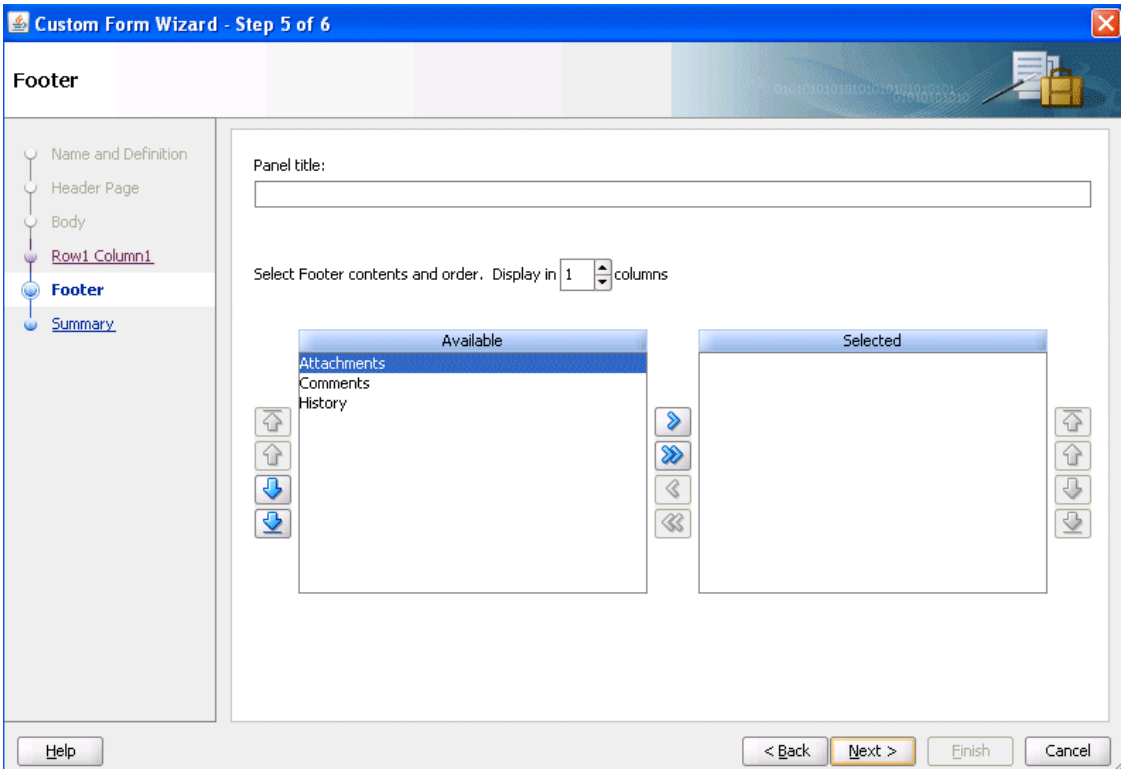
a. If you selected **Default Page Template**, the Footer page shown in Figure 28–13 is displayed. Deselect any comments, attachments, or history facet that you do not want to include in the footer, and click **Next**. By default, the comments, attachments, and history facets are all selected.

Figure 28–13 Custom Task Form Wizard: Selecting the Footer Fields for the Default Page Template



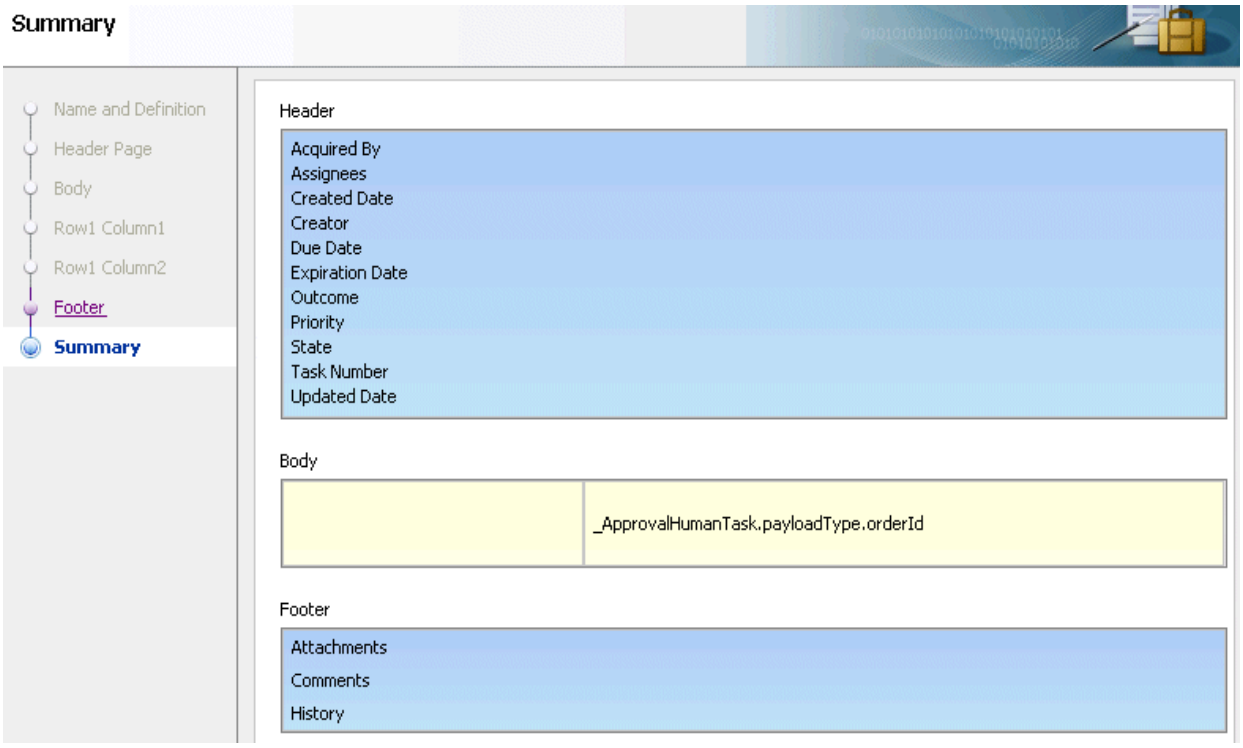
b. If you selected **Custom Page Template**, the Footer page shown in Figure 28–14 is displayed. Select any comments, attachments, or history facet that you want to move to the **Selected** list, and click **Next**.

Figure 28–14 Custom Task Form Wizard: Selecting the Footer Fields for the Custom Page Template



12. On the Summary page, shown in Figure 28–15, inspect your selections. Click **Back** to make changes or click **Finish**.

Figure 28–15 Custom Task Form Wizard: Summary



The Designer initializes and the *form_name*.jspx tab is displayed, as shown in Figure 28–16 (upper part of tab) and Figure 28–17 (lower part of tab).

Figure 28–16 Custom Task Form (Upper Part of Tab)

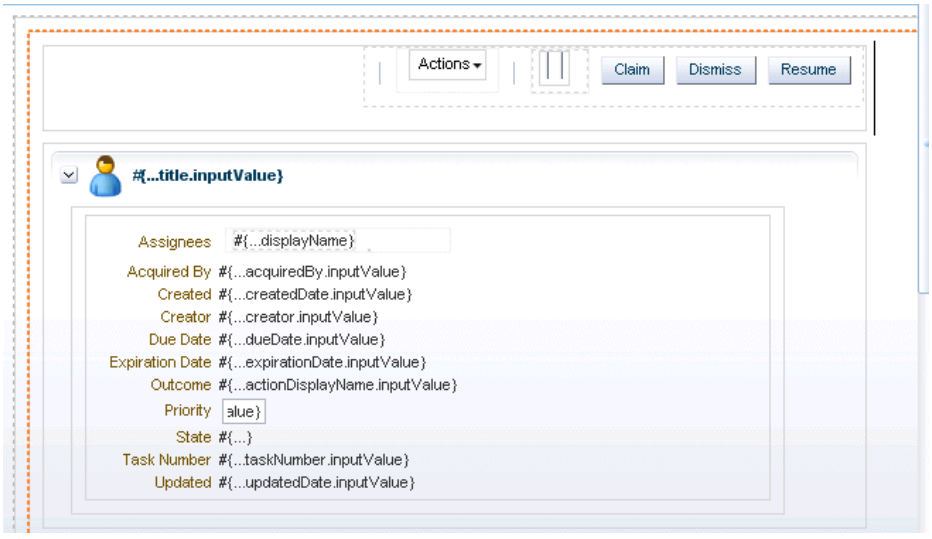
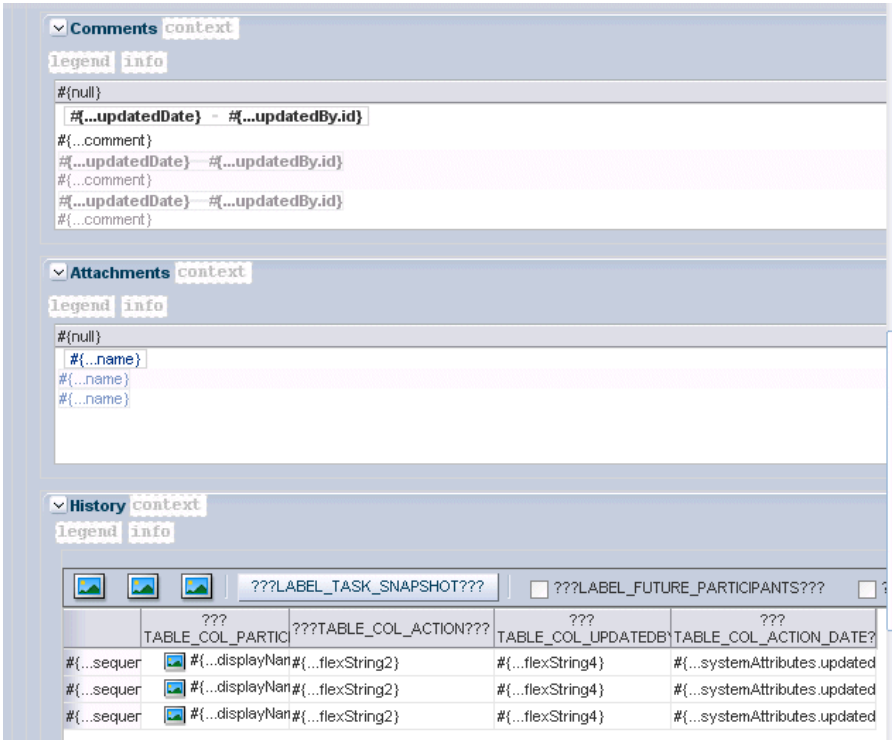


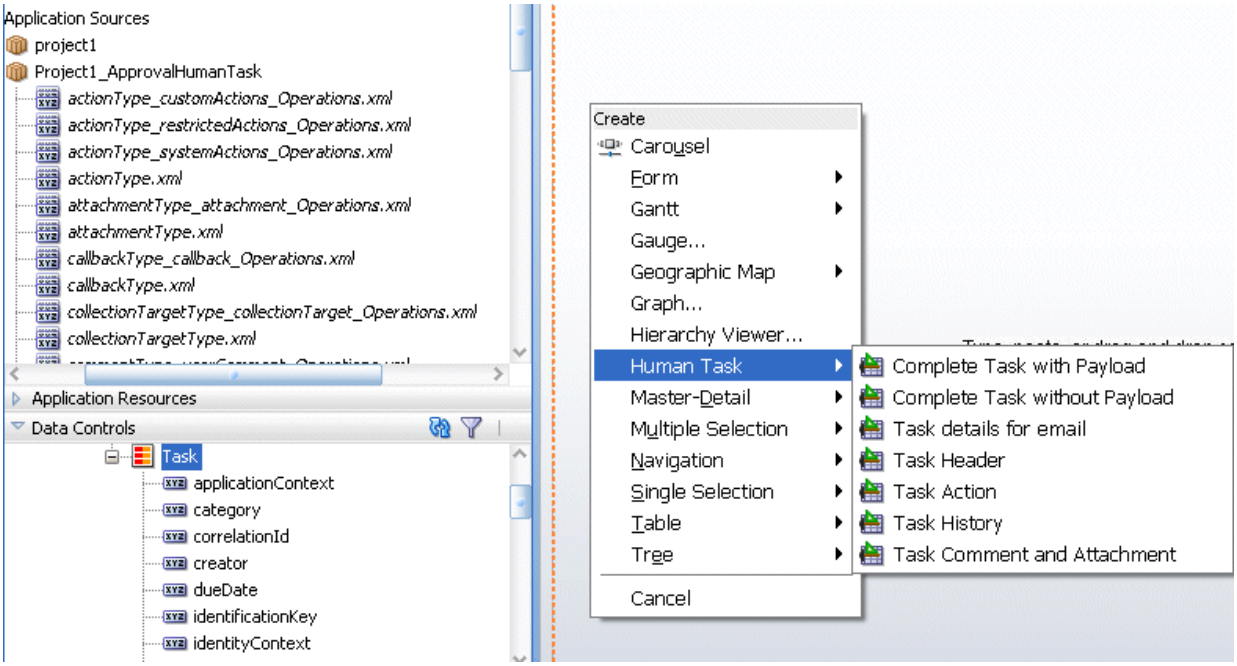
Figure 28–17 Custom Task Form (Lower Part of Tab)



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 28–18.

Figure 28–18 Human Task Drop Handlers for Creating the Task Form



Other ADF drop handlers—for forms, tables, trees, and so on (shown in Figure 28–18)—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.

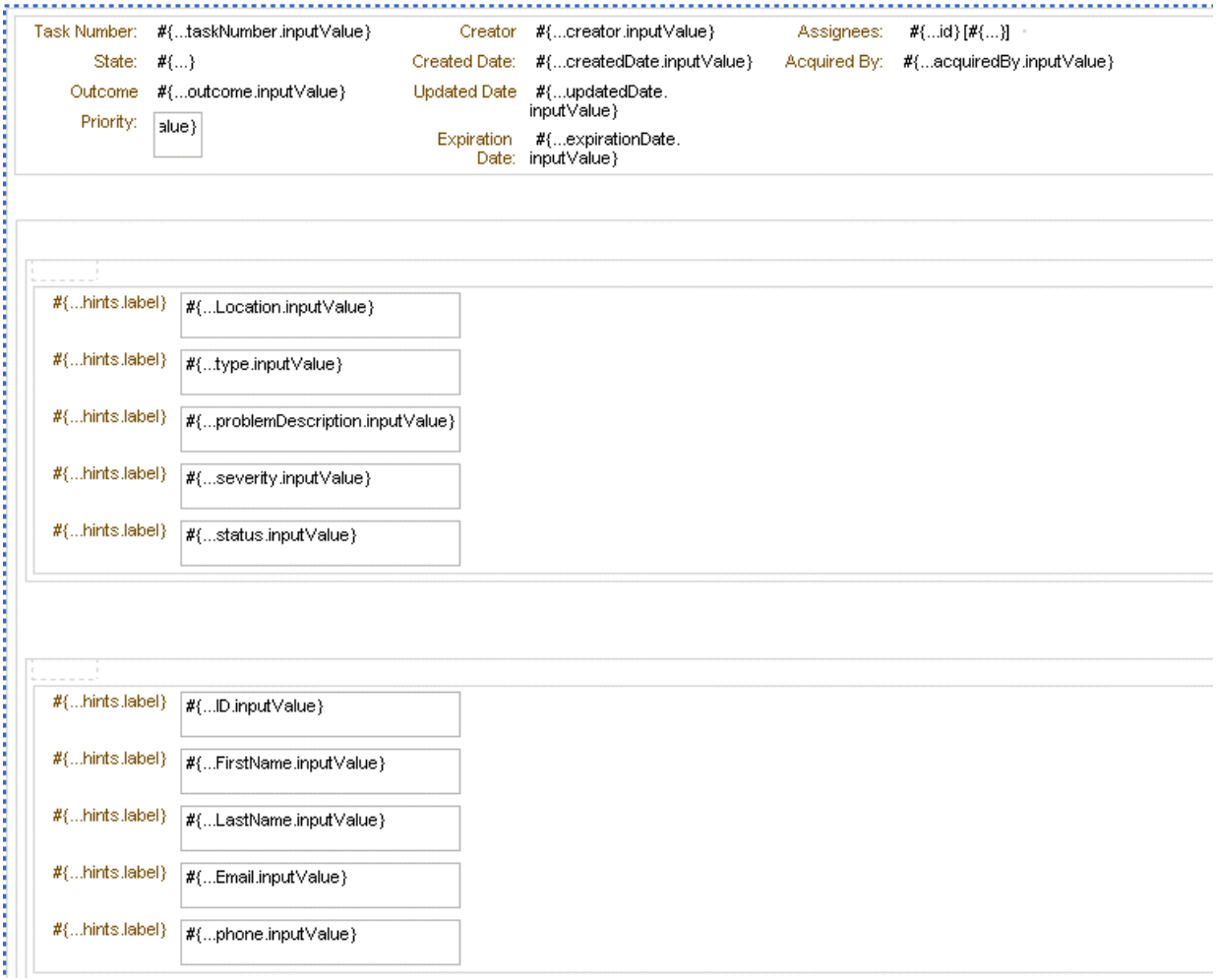
Complete Task without Payload

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 28–19.

Figure 28–19 Task Form for Email Notification



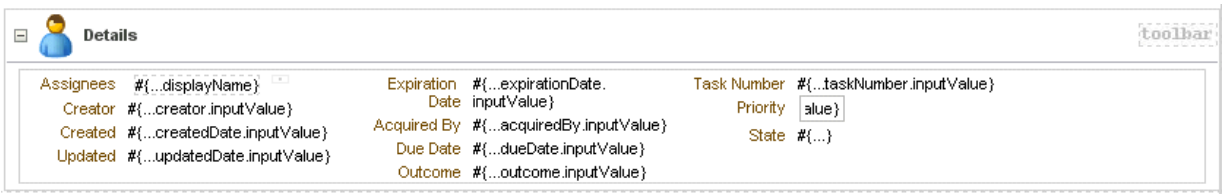
See Section 28.7, "Creating an Email Notification," for more information.

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.

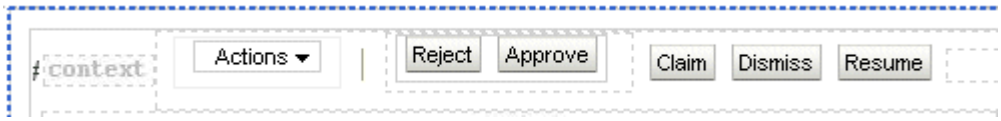
Figure 28–20 shows an example of header information.

Figure 28–20 Header Information



Buttons for task actions are also created in the header, as shown in Figure 28–21.

Figure 28–21 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. 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.

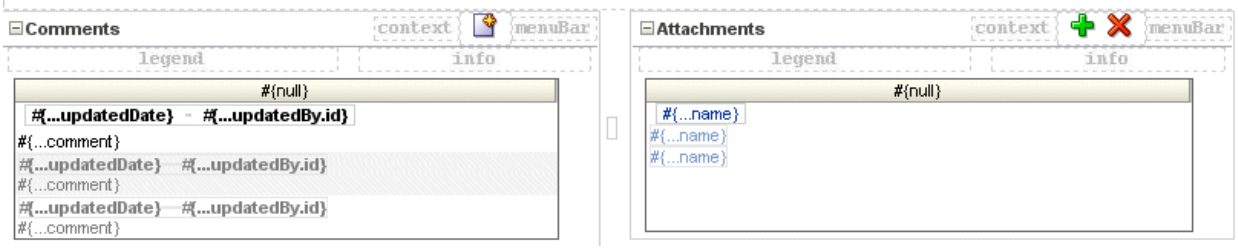
Task Comments and Attachments

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 28–22 shows an example of the comments and attachments region.

Figure 28–22 Comments and Attachment Region

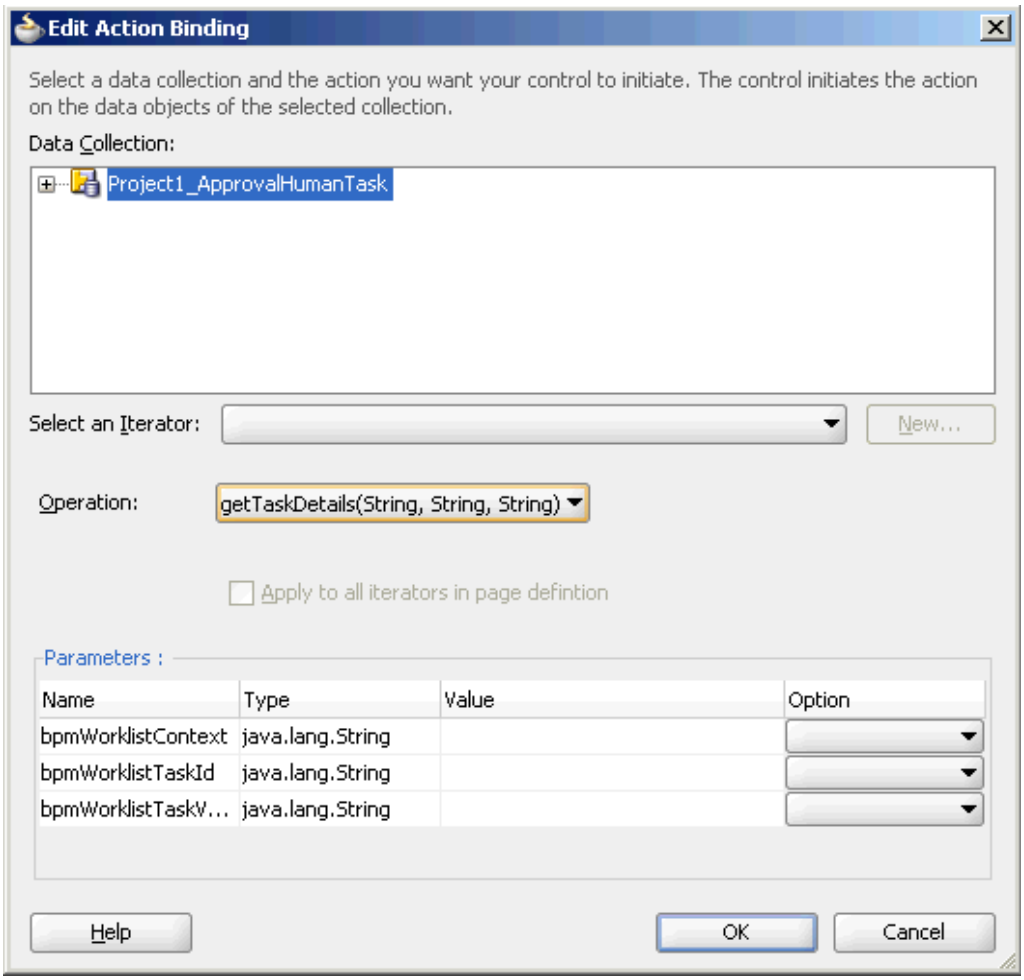


The following steps describe how to use a drop handler that creates the task form, including the payload, without building each region individually. To build each region individually, see Section 28.4.5, "How To Create Task Form Regions Using Individual Drop Handlers."

To create a task form using the Complete Task with Payload drop handler:

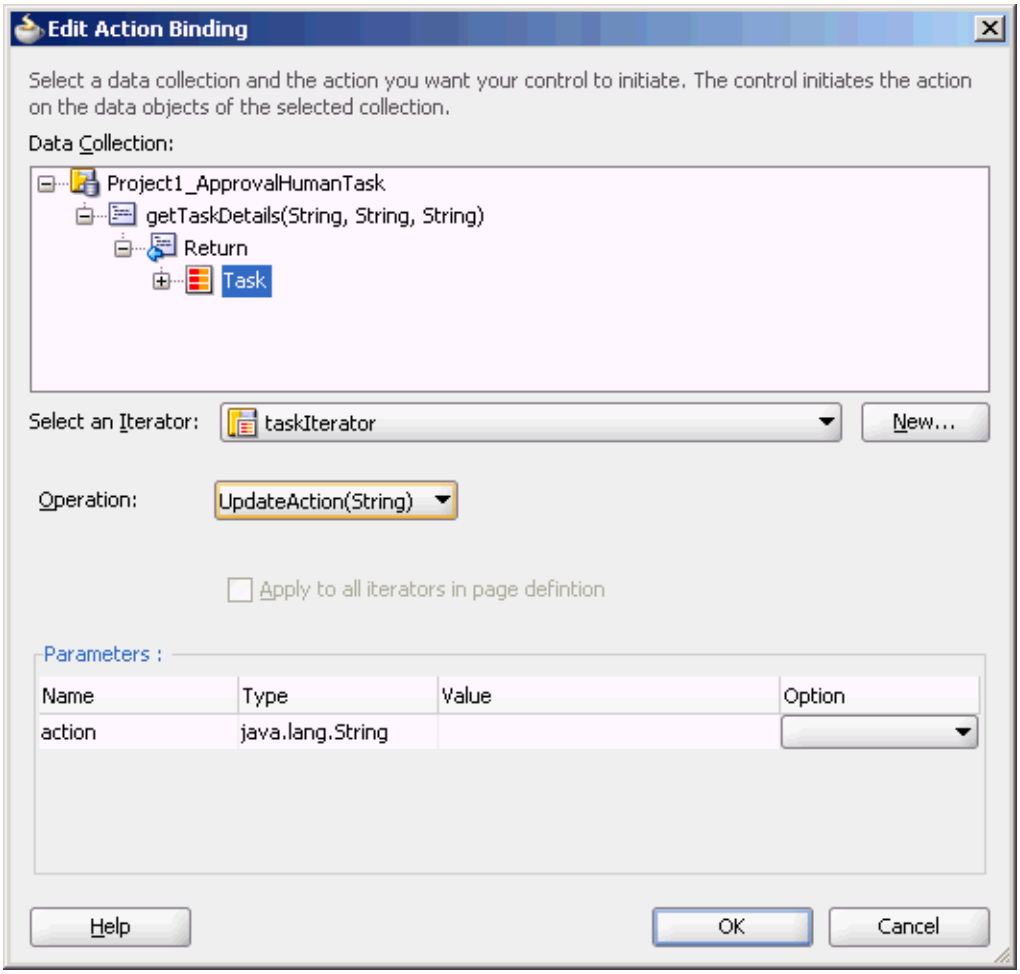
1. In the designer, double-click **taskDetails1_jspx**.
2. In the Create JSF Page dialog, provide a file name and directory information (or accept the defaults) and click **OK**.
3. In the **Data Controls** panel of the Application Navigator, expand the human task node, then the **getTaskDetails** node, 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 28–23, click **OK**.

Figure 28–23 Edit the Action Binding



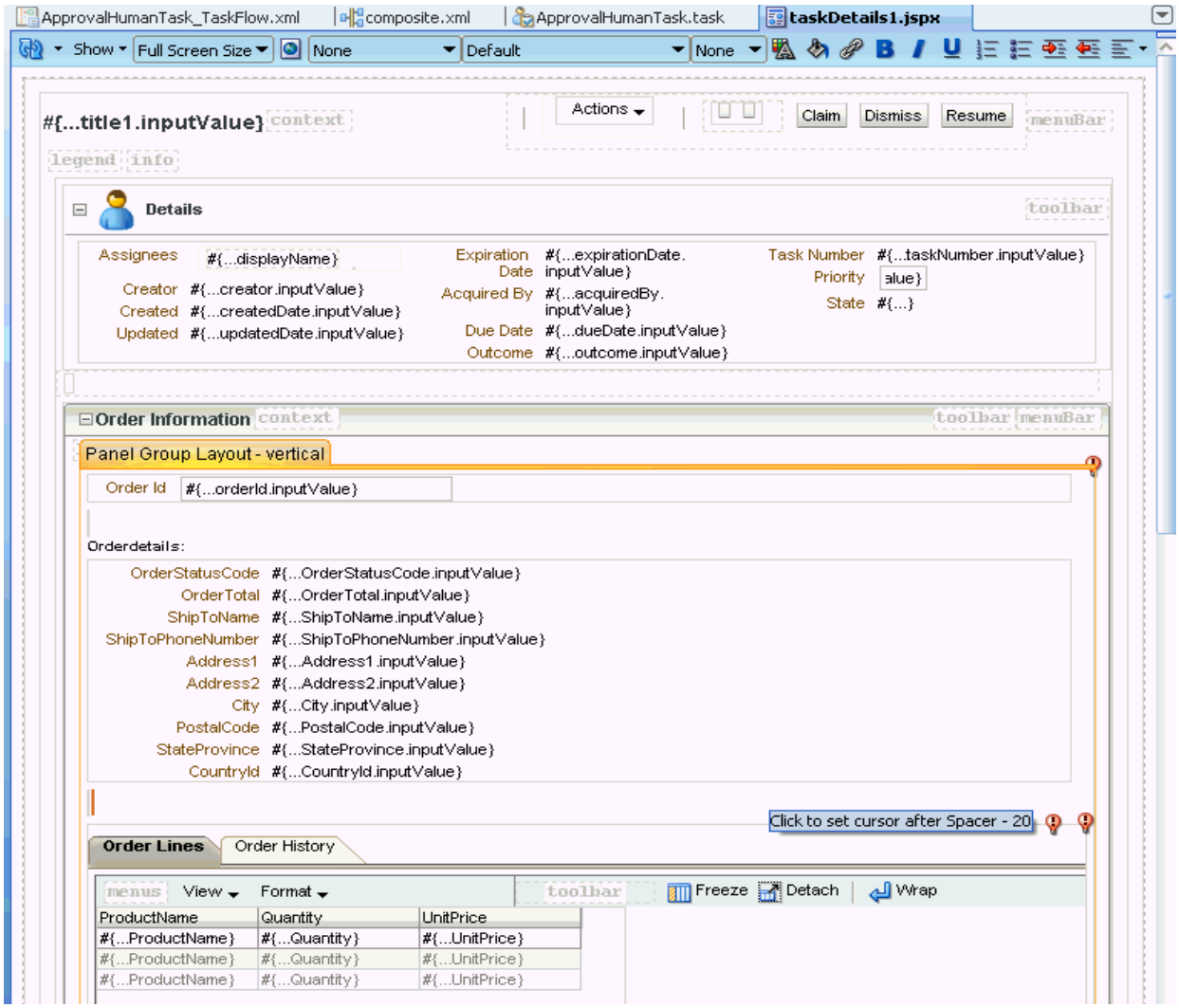
7. In the next Edit Action Binding dialog, the data collection is selected, as shown in Figure 28–24; click **OK**.

Figure 28–24 Select the Data Collection and Action



The task form is displayed, as shown in Figure 28–25.

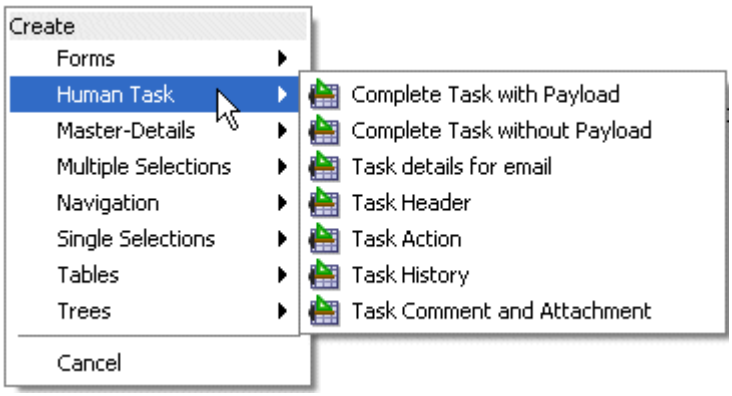
Figure 28–25 Task Form



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 28–26.

Figure 28–26 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.

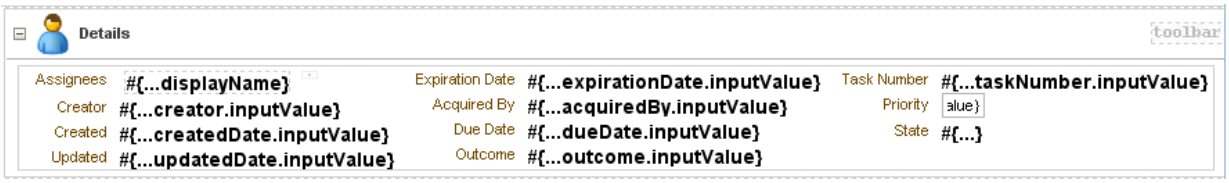
To create task form regions using individual drop handlers:

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 **taskDetails.jspx** window.
5. Select **Human Task**, and then **Task Header**.

   This creates the **Actions** dropdown list and buttons for task actions, as shown in Figure 28–27, and header details, as shown in Figure 28–28.

Figure 28–27 Designing the Task Form: Buttons for Task Actions

Figure 28–28 Designing the Task Form: Task Headers



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

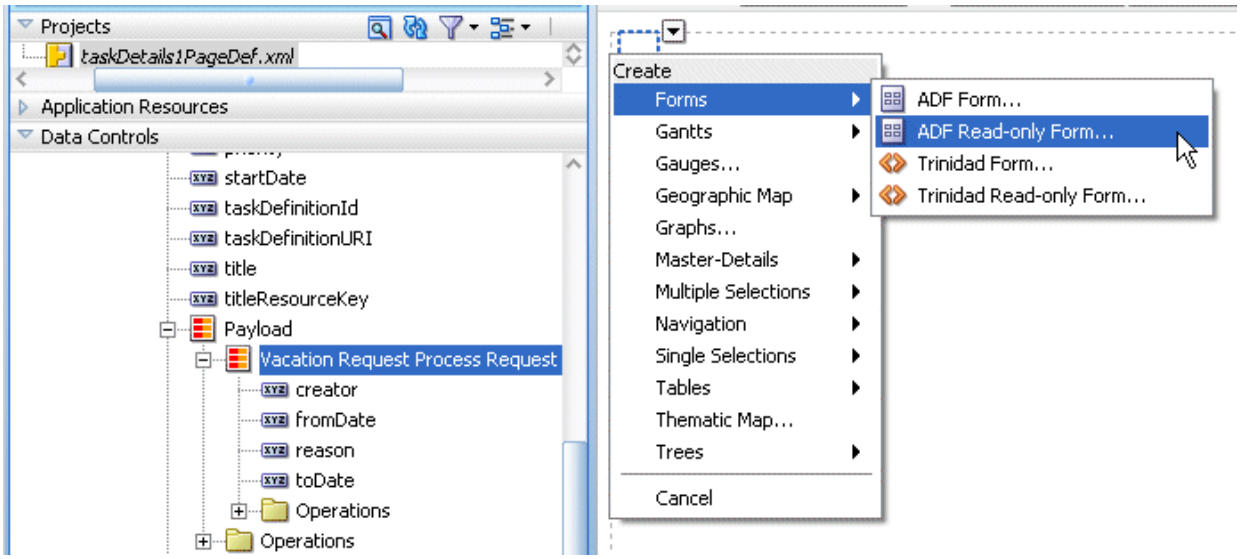
In addition to adding the payload, you can create task form regions. See Step 1 in Section 28.4.5, "How To Create Task Form Regions Using Individual Drop Handlers."

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 28–29.

Figure 28–29 Adding ADF Read-Only Fields to the Task Form Payload Region

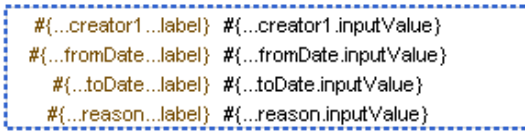


7. In the Edit Form Fields dialog, accept the defaults and click **OK**.

   This creates read-only fields in the payload region, between the **Details** and **History** sections.

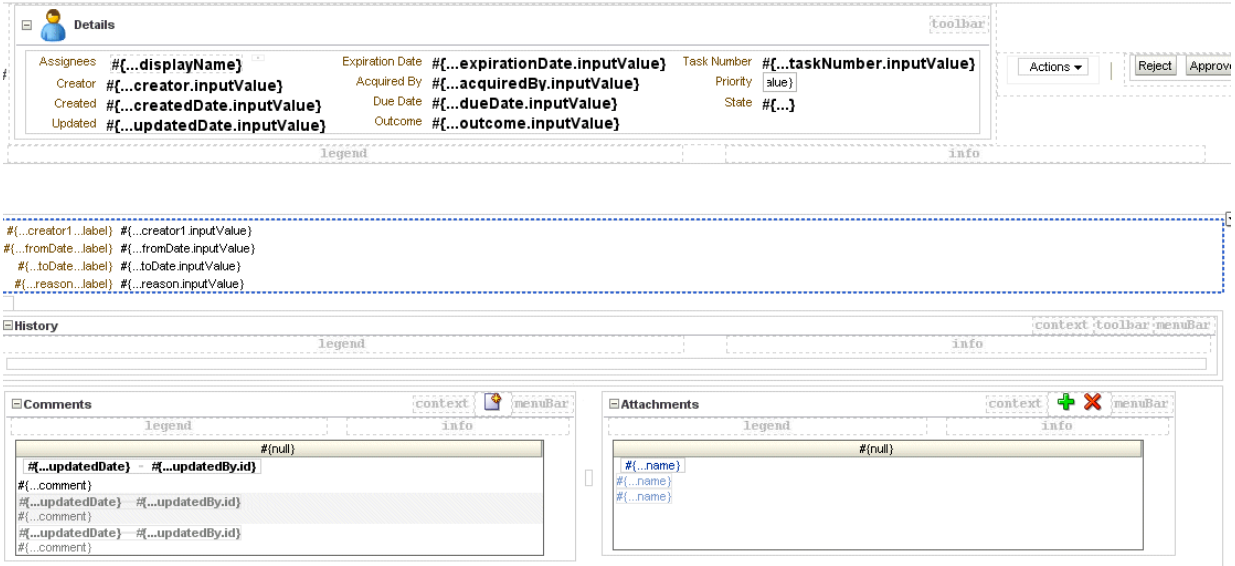
   The payload regions appear, as shown in Figure 28–30.

Figure 28–30 The Payload Region of the Task Form



The task form, shown in Figure 28–31, is complete and ready to be deployed.

Figure 28–31 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

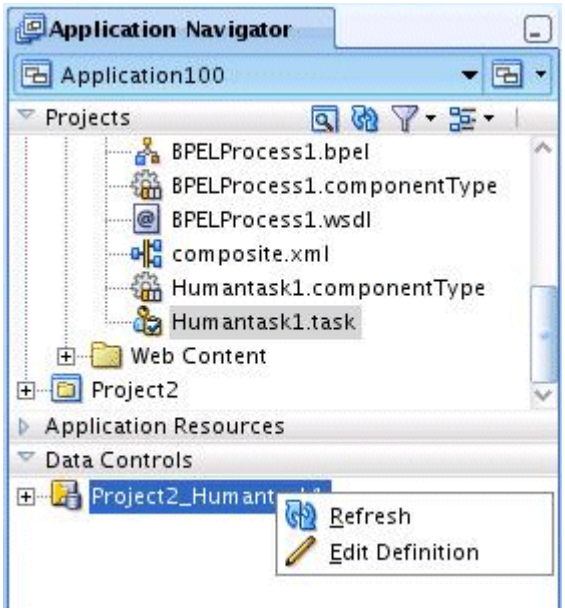
JDev_Oracle_Home`\mywork\`*task_form_application_name*`\`*project_name*`\public_html`

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 28–32 shows the **Refresh** option.

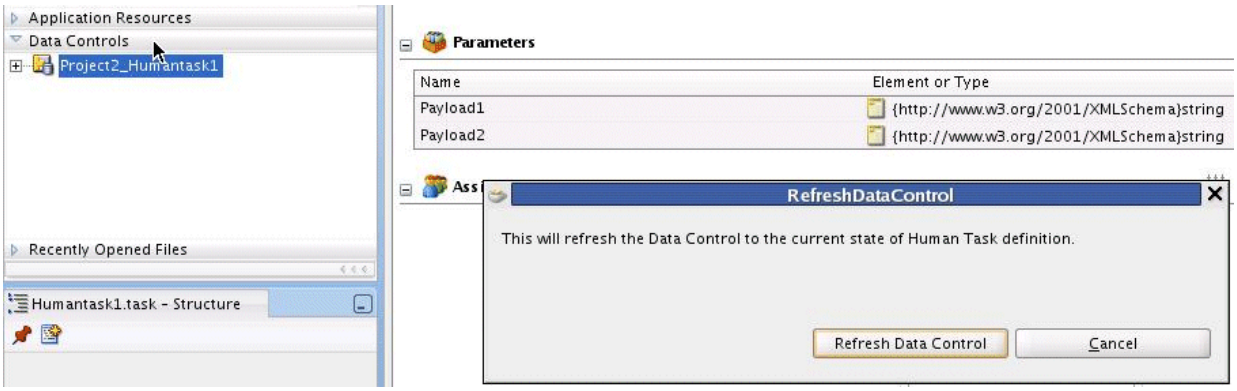
Figure 28–32 Refreshing Data Controls



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 28–33.

Figure 28–33 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**.

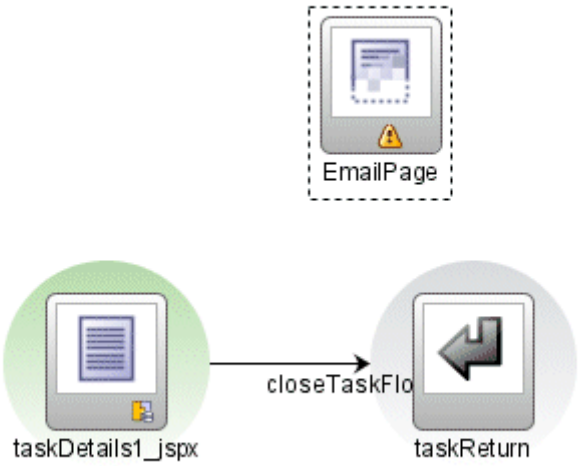
   The XML file opens in the designer. In the diagram view, you see the **taskDetails1.jspx** icon.

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 28–34 shows an example using the name **EmailPage**.

Figure 28–34 Creating the Email Page



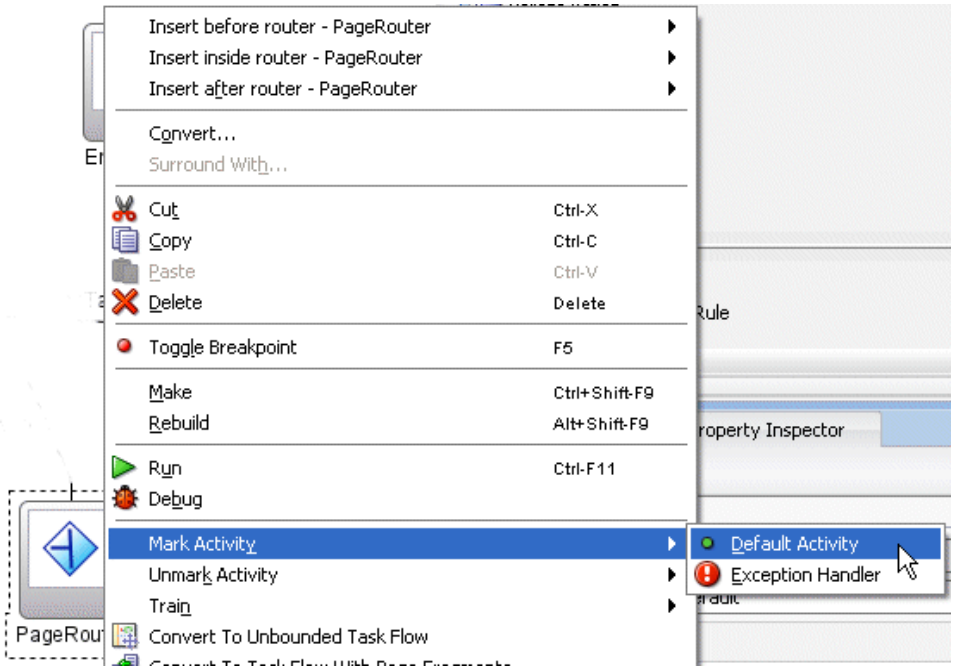
4. From the **Component Palette**, drag **Router** into the designer.

5. Click **router1** below the icon and enter a router name.

   Figure 28–36 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 28–35.

Figure 28–35 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.

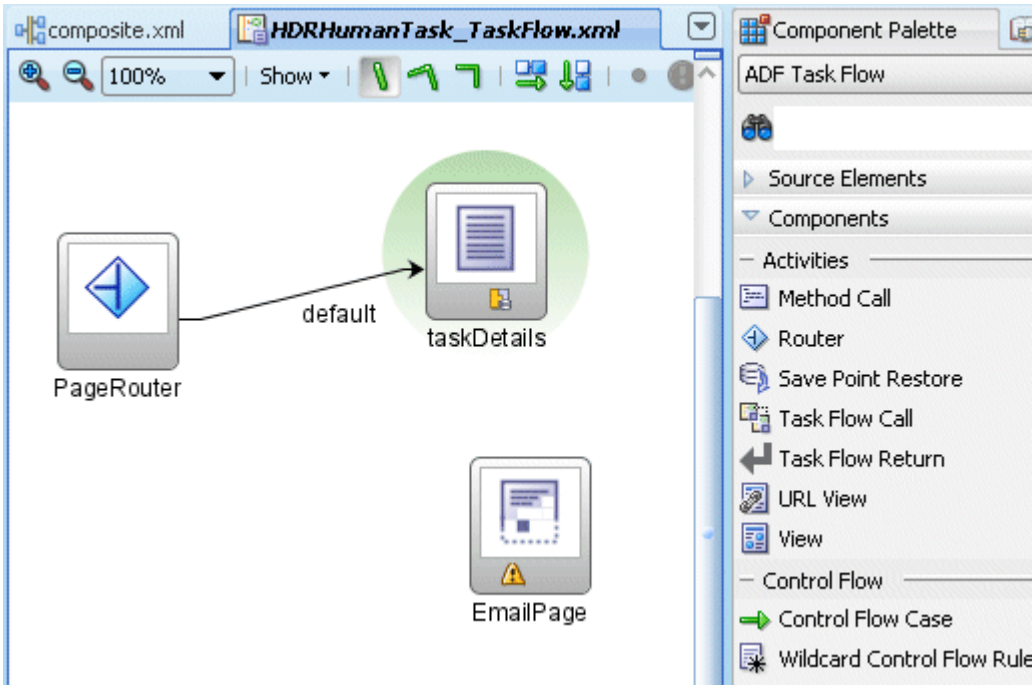
10. Use the Expression Builder to enter the following in the **expression** field:
    `#{pageFlowScope.bpmClientType=="notificationClient"}`

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 28–36.

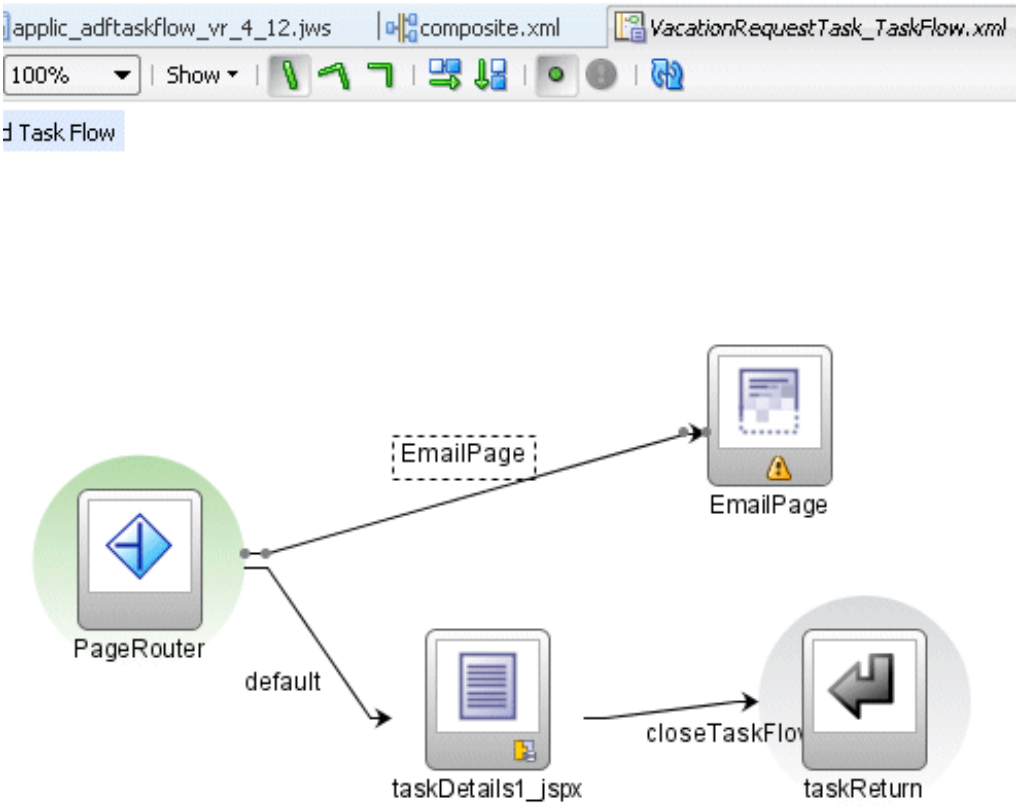
Figure 28–36 Connecting the Control Flow



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 28–37 shows the completed control flow.

Figure 28–37 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.

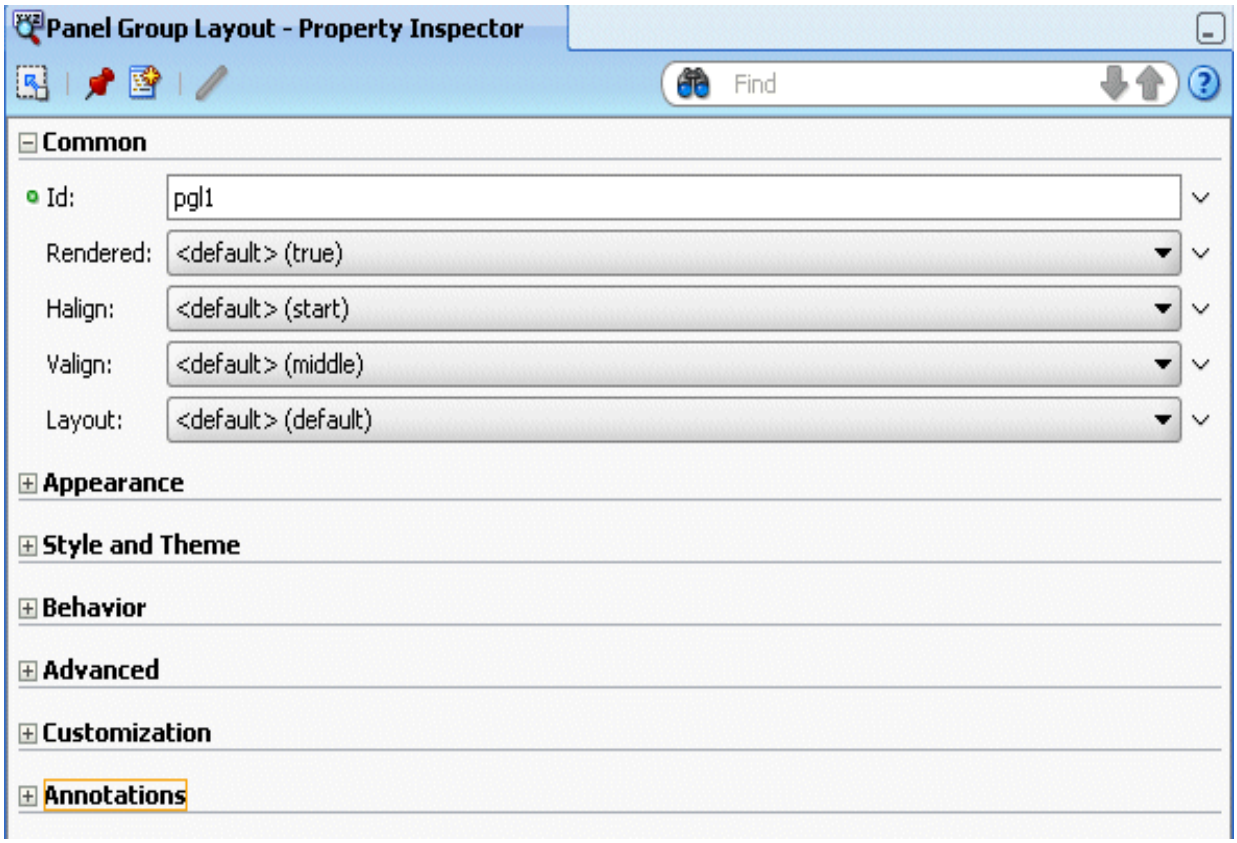
To create an email notification page:

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 28–38 shows the layout fields available when **Panel Group Layout** is selected.

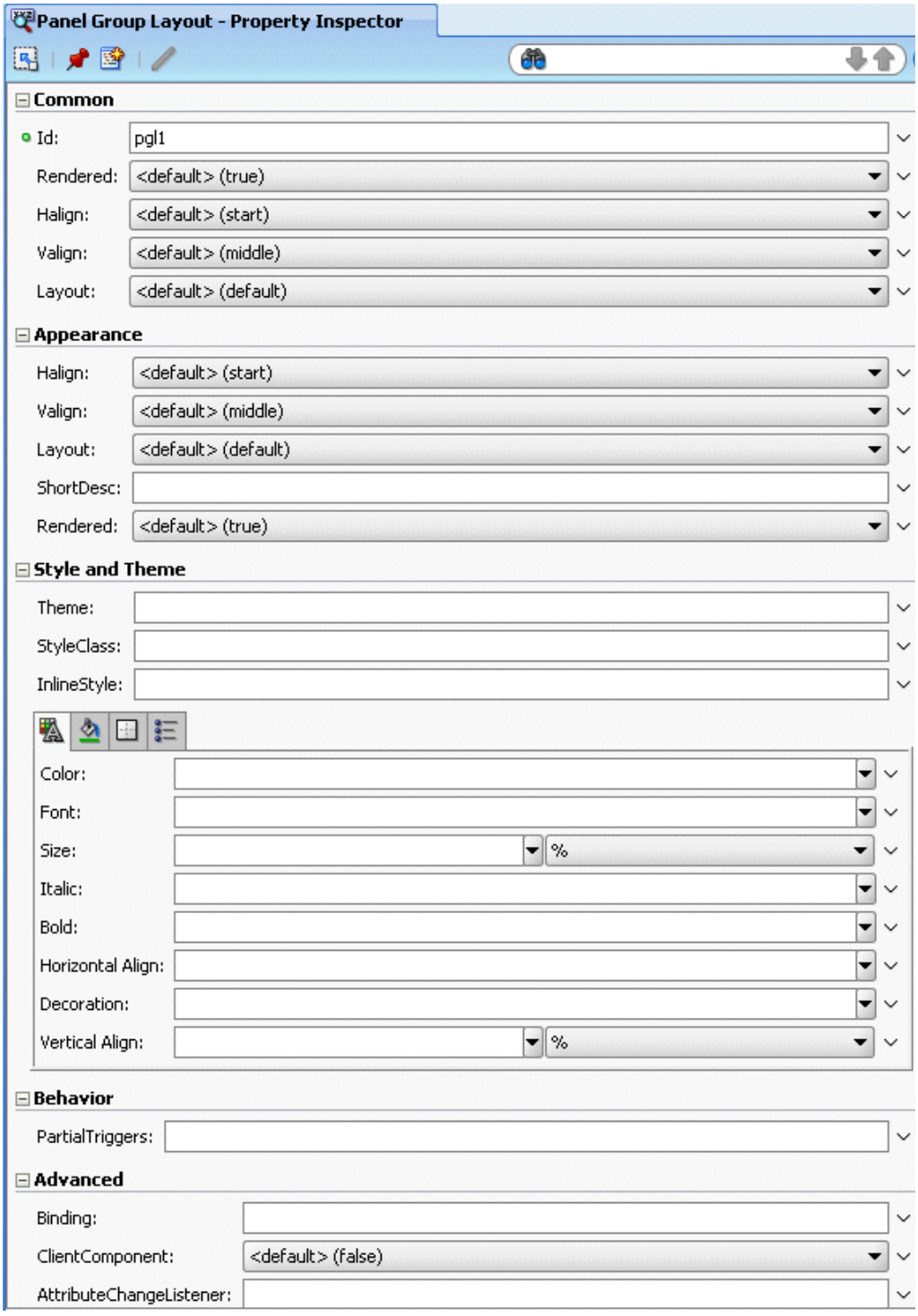
Figure 28–38 Specifying a Layout



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 28–39.

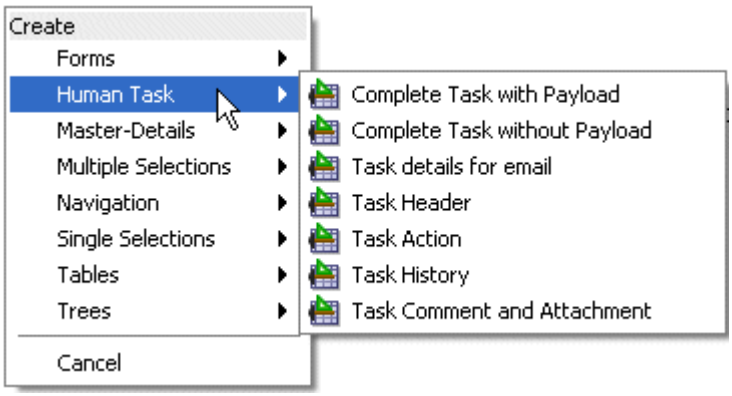
Figure 28–39 Specifying a Layout: More Details



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 28–40.

Figure 28–40 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</initialContextFactor
y>

<participateInClientTransaction>false</participateInClientTransaction>
        </remoteClient>
```

```
        <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.

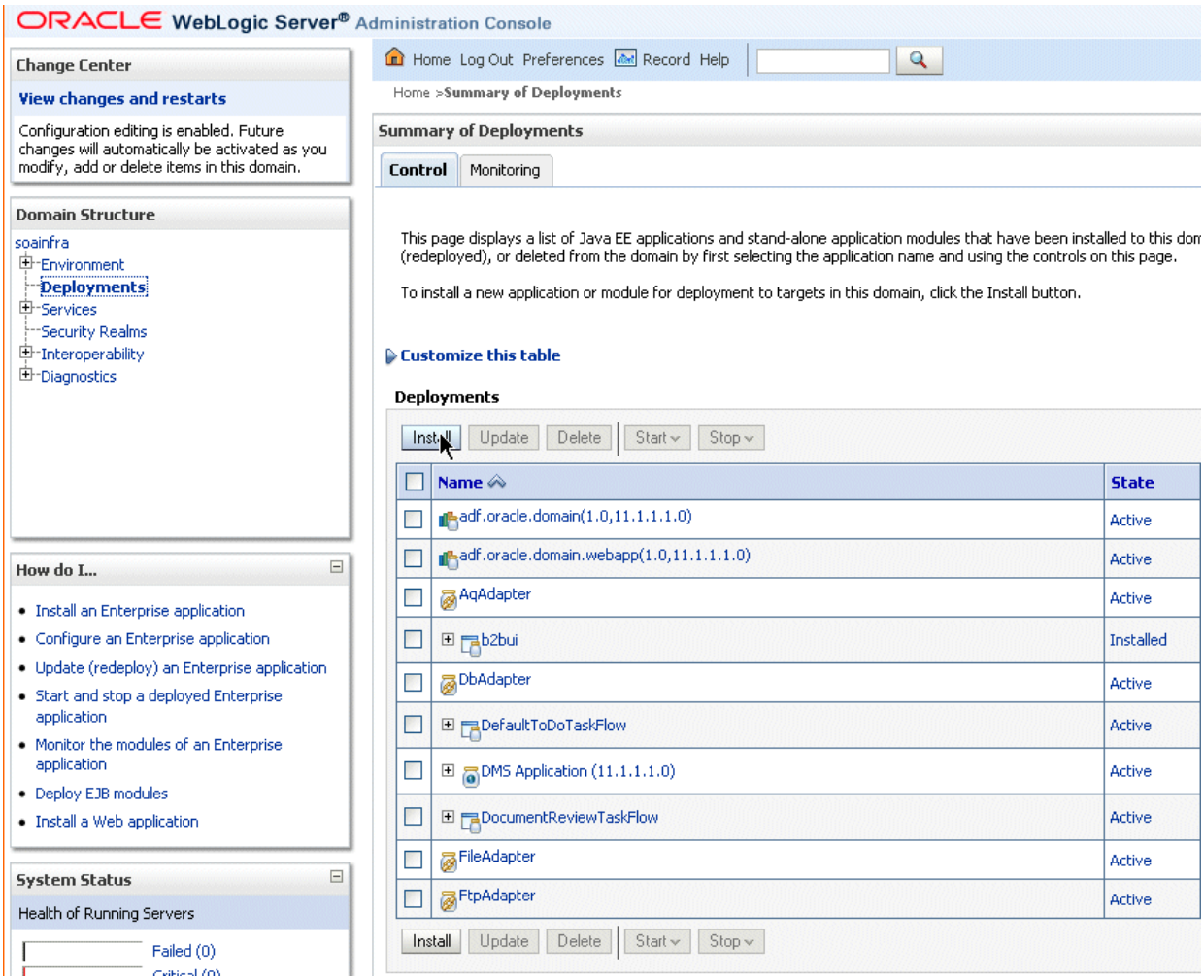
To deploy oracle.soa.workflow.jar:

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 28–41.

Figure 28–41 Oracle WebLogic Server Administration Console: List of Deployments

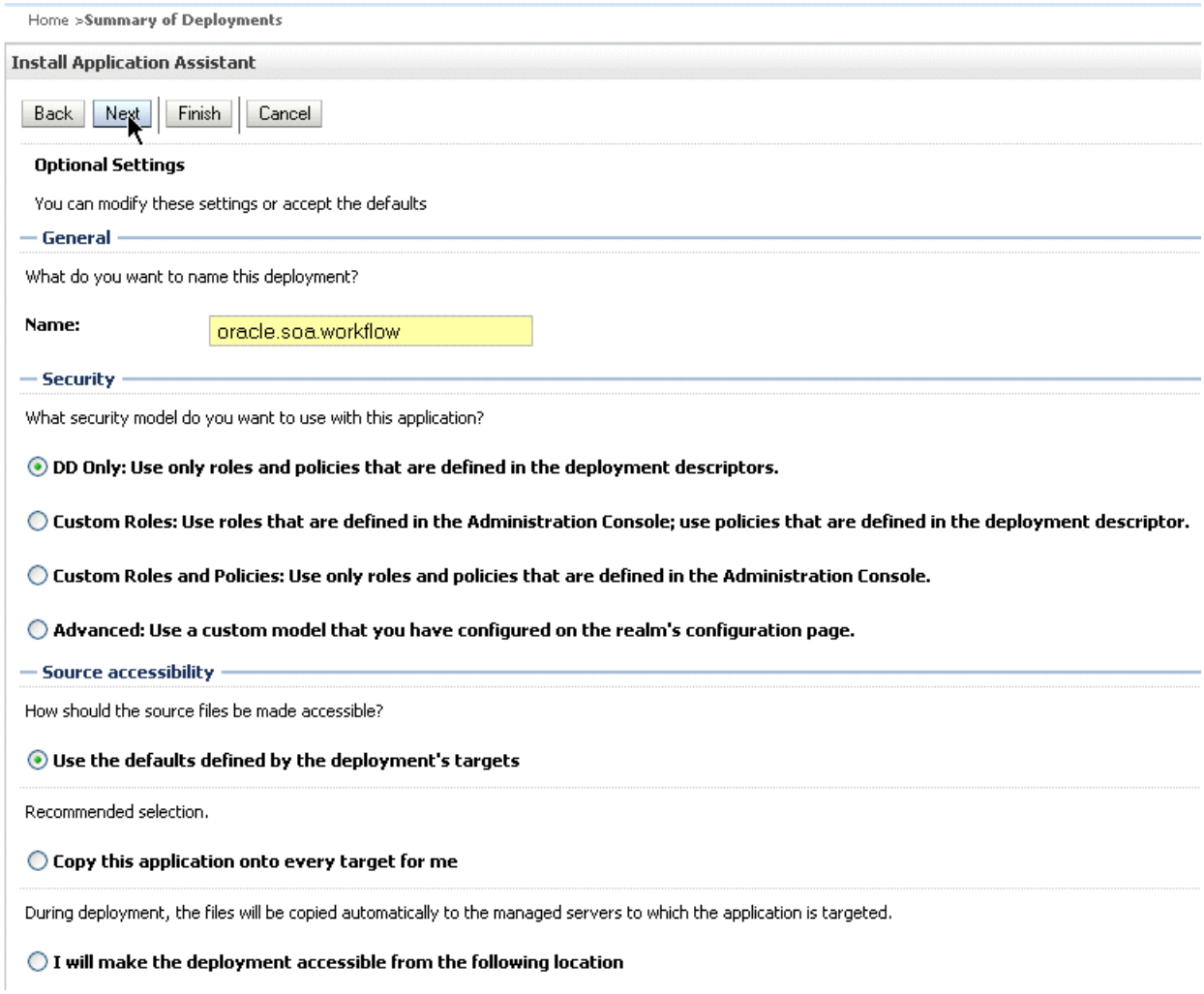


4. In the **Path** field, provide the following path and click **Next**.

   ORACLE_JDEV_HOME`/jdeveloper/soa/modules/oracle.soa.workflow_11.1.1/oracle.soa.workflow.jar`

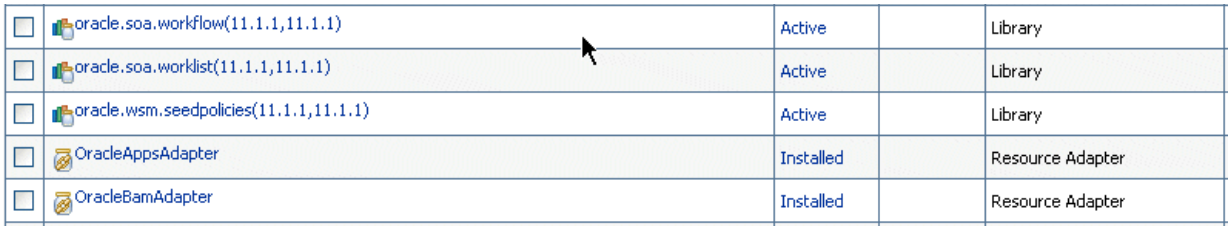
5. Keep the same name for the deployment and click **Next**, as shown in Figure 28–42.

Figure 28–42 Oracle WebLogic Server Administration Console: Install Applications Assistant



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 28–43.

Figure 28–43 Oracle WebLogic Server Administration Console: The oracle.soa.workflow Active State



See Section 28.8.4.4, "Defining the Foreign JNDI Provider on a non-SOA Oracle WebLogic Server," to continue.

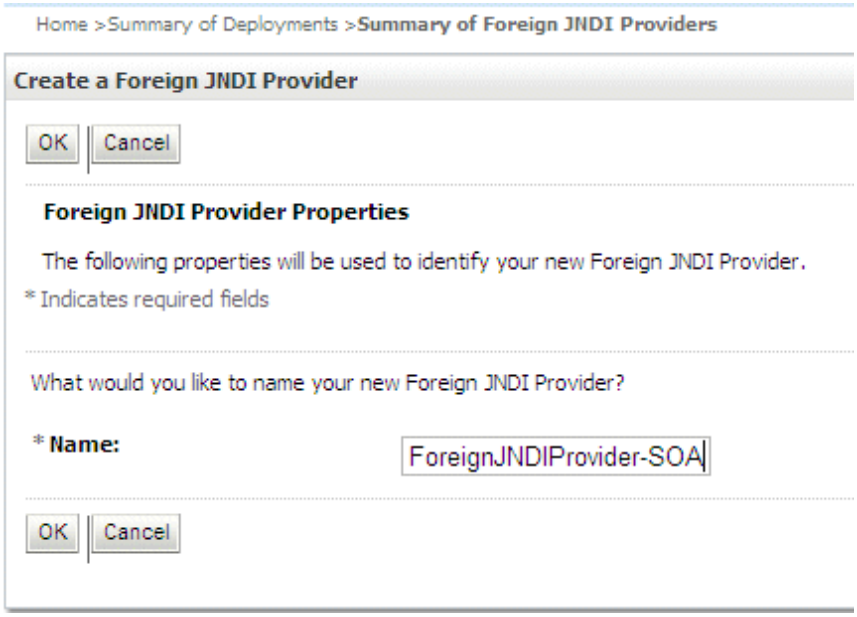
28.8.4.4 Defining the Foreign JNDI Provider on a non-SOA Oracle WebLogic Server

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 28–44, and click **OK**.

Figure 28–44 Creating a Foreign JNDI Provider

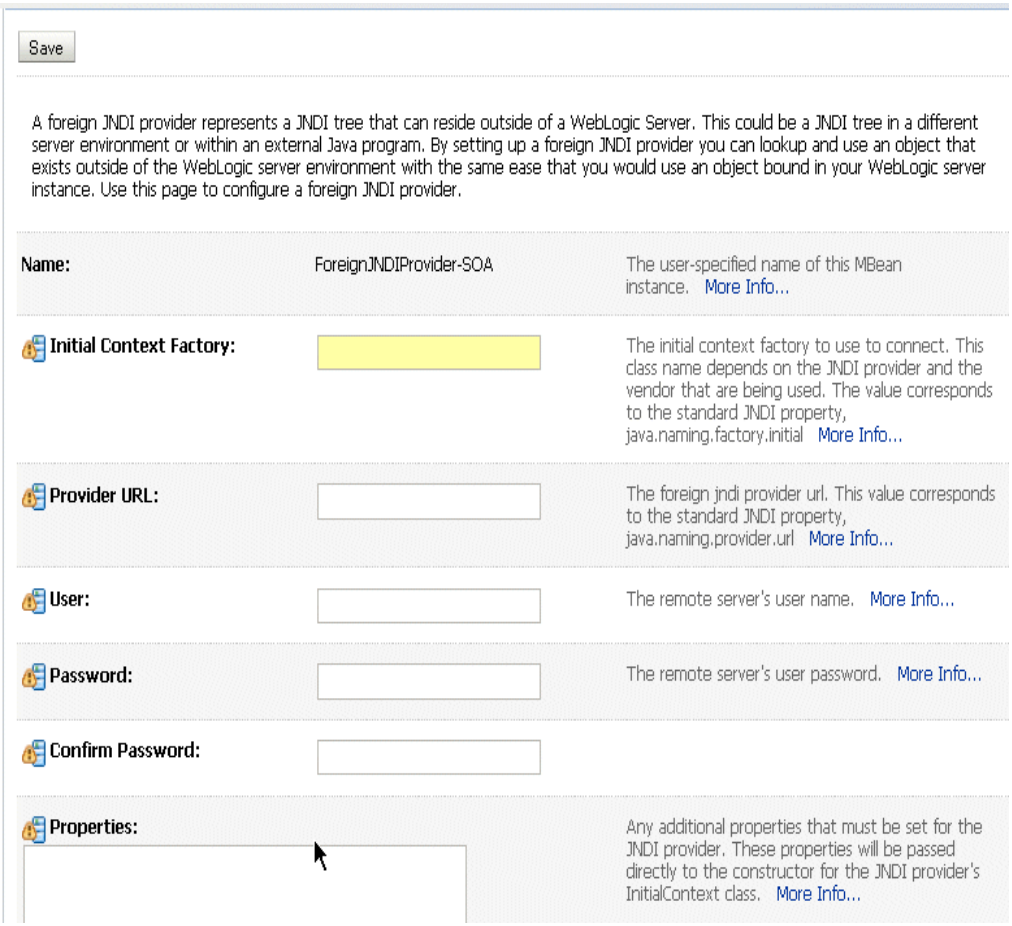


4. Click the **ForeignJNDIProvider-SOA** link.
5. Do the following and click **Save**.
   - For **Initial Context Factory**, enter `weblogic.jndi.WLInitialContextFactory`.
   - For **Provider URL**, enter `t3://`*`soa_hostname`*`:`*`soa_portnumber`*`/soa-infra`.

     In a clustered environment, for **Provider URL**, enter `http://`*`soa_hostname`*`:`*`soa_portnumber`*`/soa-infra`.
   - For **User**, enter `weblogic`.
   - For **Password**, enter `weblogic`.

   Figure 28–45 shows the page where you enter this information.

Figure 28–45 Defining the Foreign JNDI Provider



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

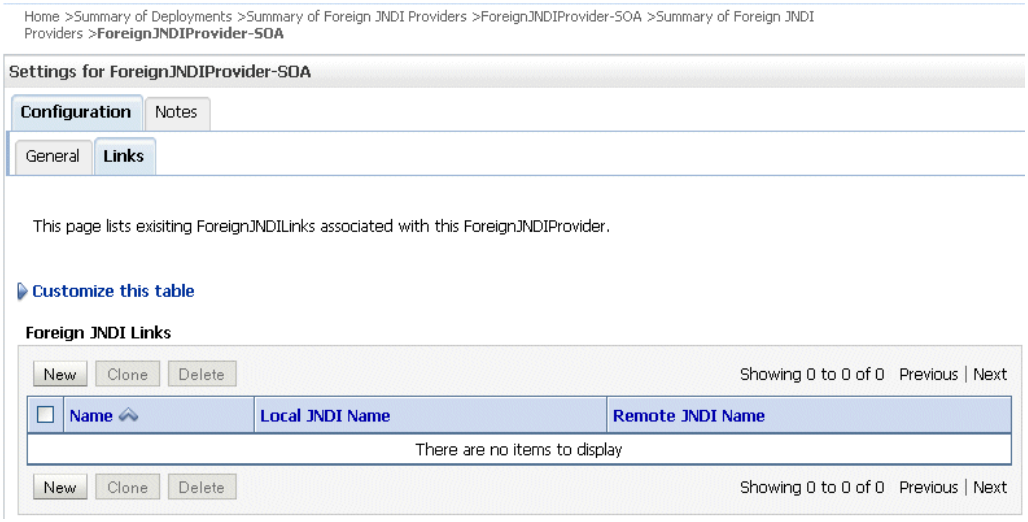
Use Oracle WebLogic Server Administration Console to complete this portion of the task.

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 28–46 shows the **Links** tab.

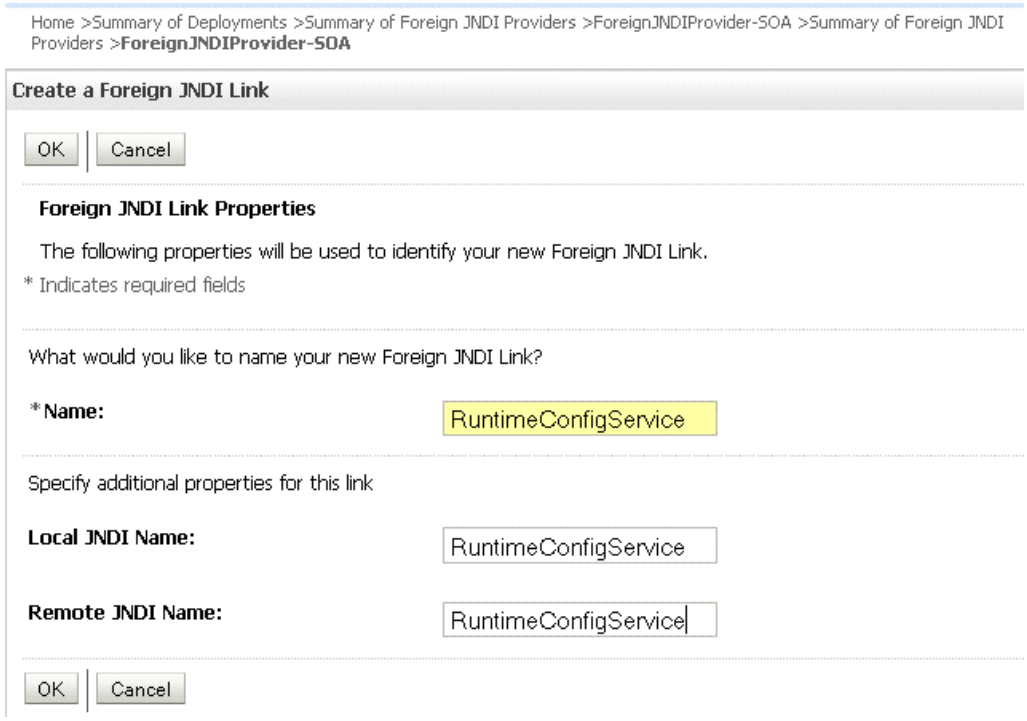
Figure 28–46 Defining the Foreign JNDI Provider Links: The Links Tab



5. Do the following and click **OK**.
   - For **Name**, enter `RuntimeConfigService`.
   - For **Local JNDI Name**, enter `RuntimeConfigService`.
   - For **Remote JNDI Name**, enter `RuntimeConfigService`.

   Figure 28–47 shows where you do this.

Figure 28–47 Defining the Foreign JNDI Provider Links: Link Properties



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 11*g* 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

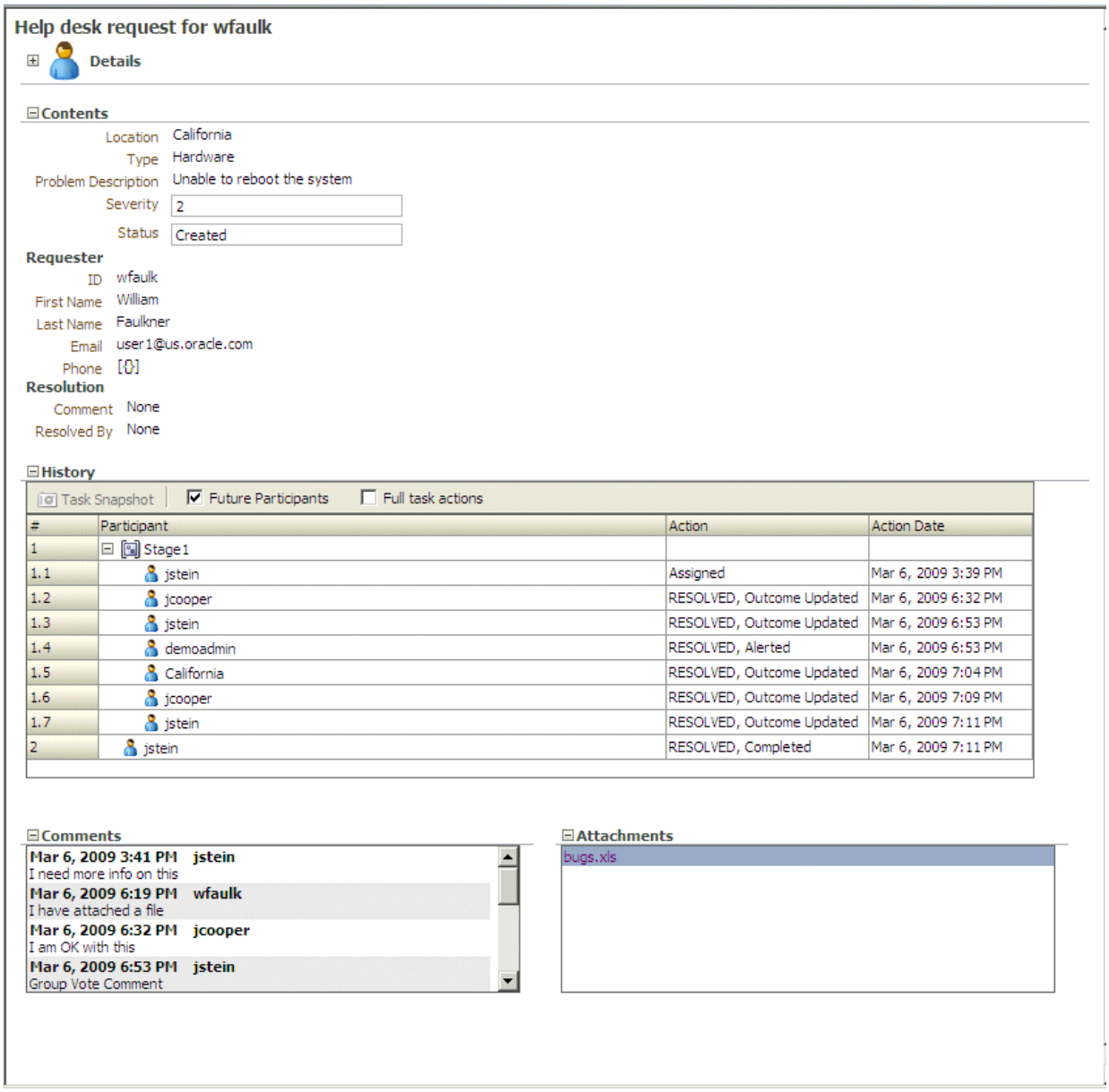
When a task flow Web application is deployed, the task flow URL is registered in the database. This URL is displayed in Oracle BPM Worklist when a task is clicked and the task details are displayed. If the task flow Web application is later undeployed or stopped, the task flow URL in the database is *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 28–48 shows how the task form for the help desk request example is displayed in the Worklist Application task details page.

Figure 28–48 Worklist Task Details Page



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 28–49 shows how an email task notification appears in email.

Figure 28–49 Email Task Notification

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 Applicationin 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.

29

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:

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.

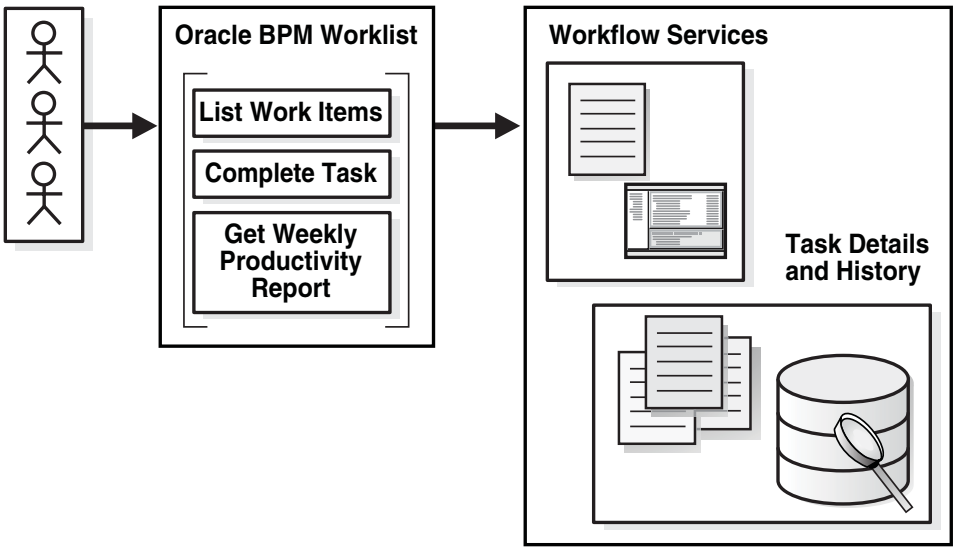
Users can customize their task lists, as required, by adding worklist views, for example, selecting the columns to display, or displaying a subset of the tasks based on filter criteria.

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 and Oracle Business Process Management 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 Set the Worklist Display (Application Preferences)," for information on how administrators can set a preference to change the realm label displayed in the interface, or specify an alternative location for the source of the login page image.

After a user logs in, the **Home** (task list) page displays tasks for the user based on the user's permissions and assigned groups and roles. The **My Tasks** tab and the **Inbox** are displayed by default. The actions allowed from the **Actions** list also depend on the logged-in user's privileges.

Figure 29–2 shows an example of the **Home** page.

Figure 29–2 Oracle BPM Worklist—The Home (Task List) Page

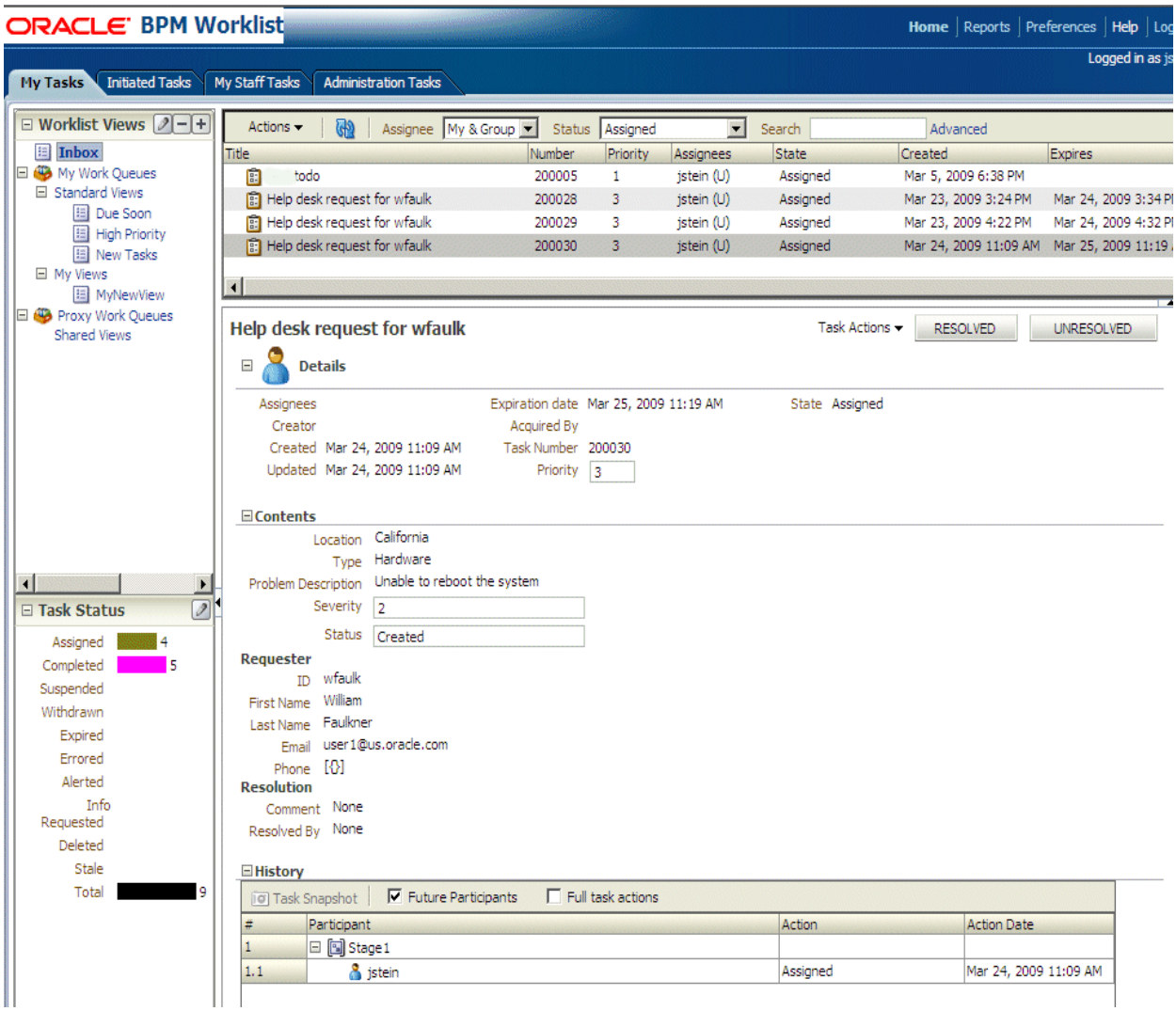


Table 29–2 describes the components of the **Home** (task list) page.

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 S**earch** 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

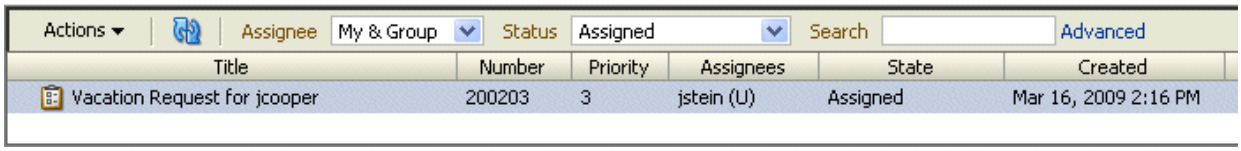
You can customize your task list in several ways, including adding worklist views, selecting which columns to display, and displaying a subset of the tasks based on filter criteria. Resize the task list display area to increase the number of tasks fetched.

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



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

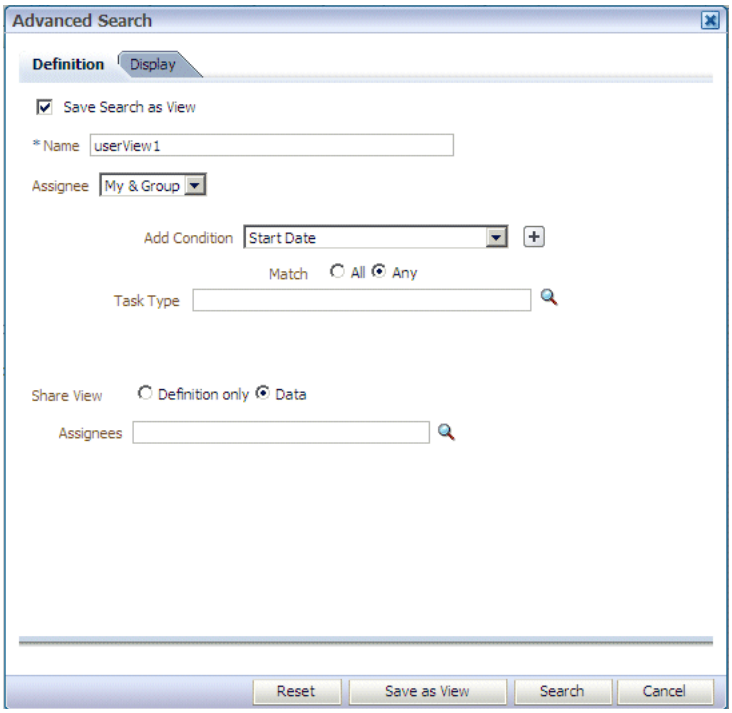


Figure 29–5 Worklist Advanced Search—Display Tab

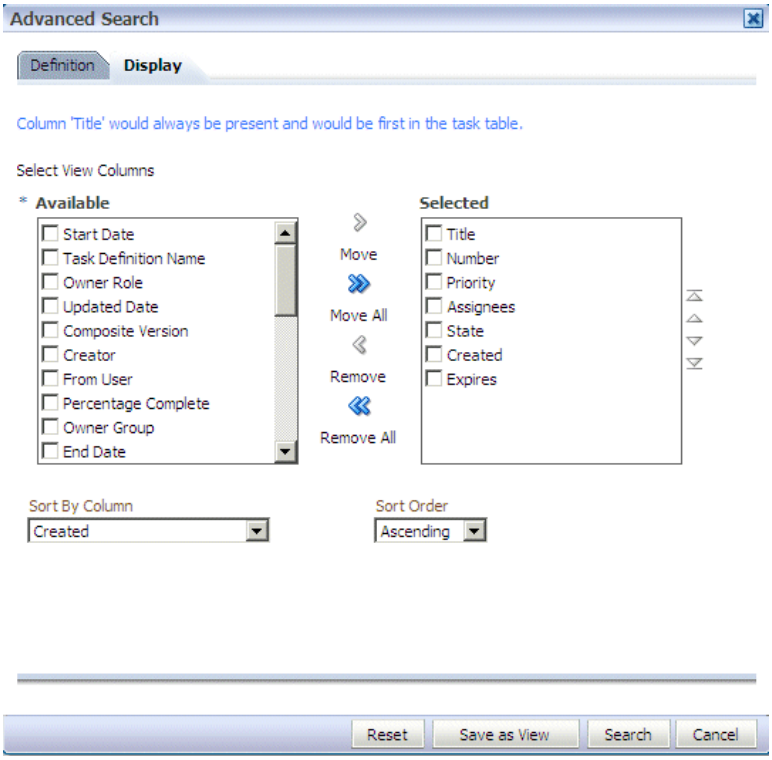


Table 29–4 describes the advanced search view columns available in the **Display** tab.

Table 29–4 Advanced Search—View Columns

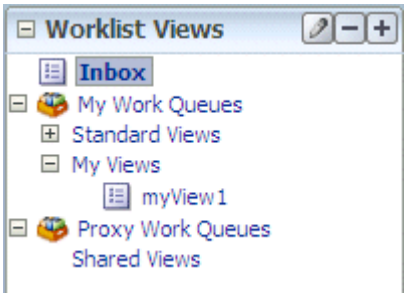
Column	Description
Start Date	The start date of the task (used with ToDo tasks).
Task Definition Name	The name of the task component that defines the task instance.
Owner Role	The application role (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner 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.

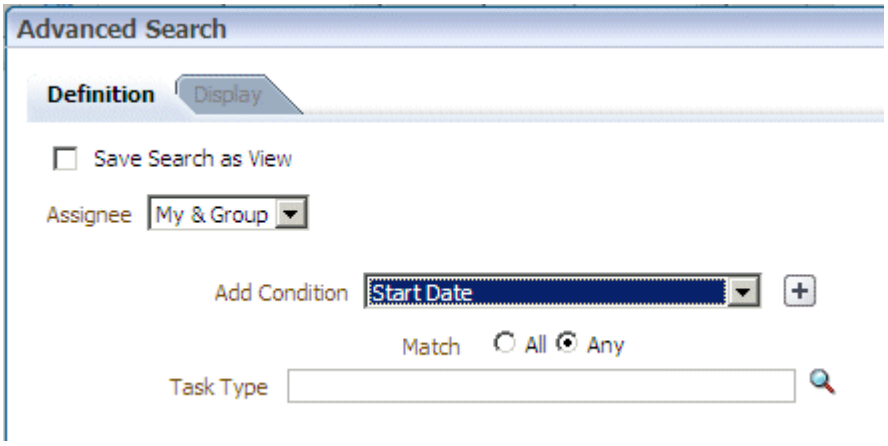
The saved view appears in the **Inbox** under **My Views**, as shown in Figure 29–6.

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

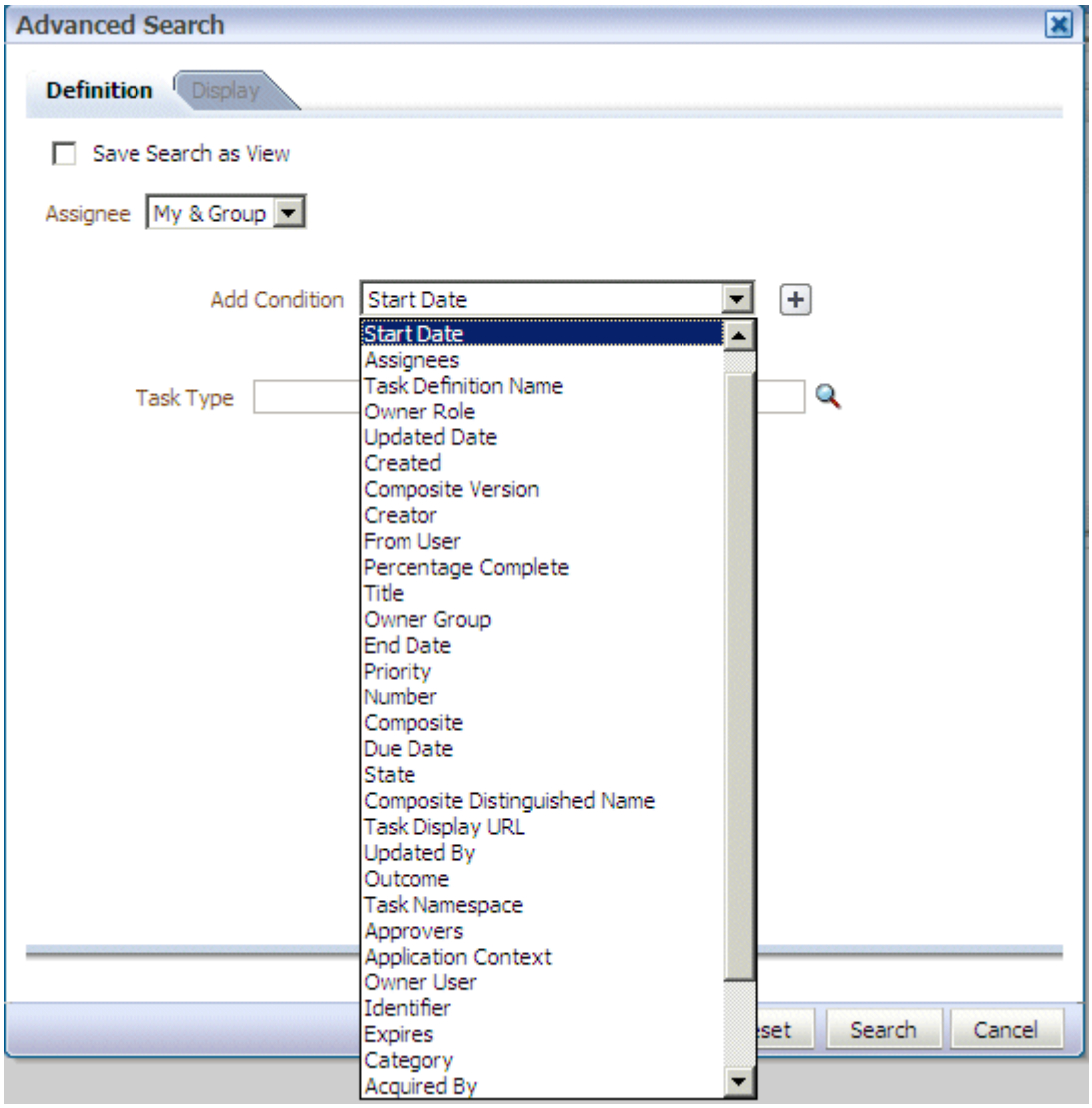


Table 29–5 describes the available conditions.

Table 29–5 Advanced Search—Conditions

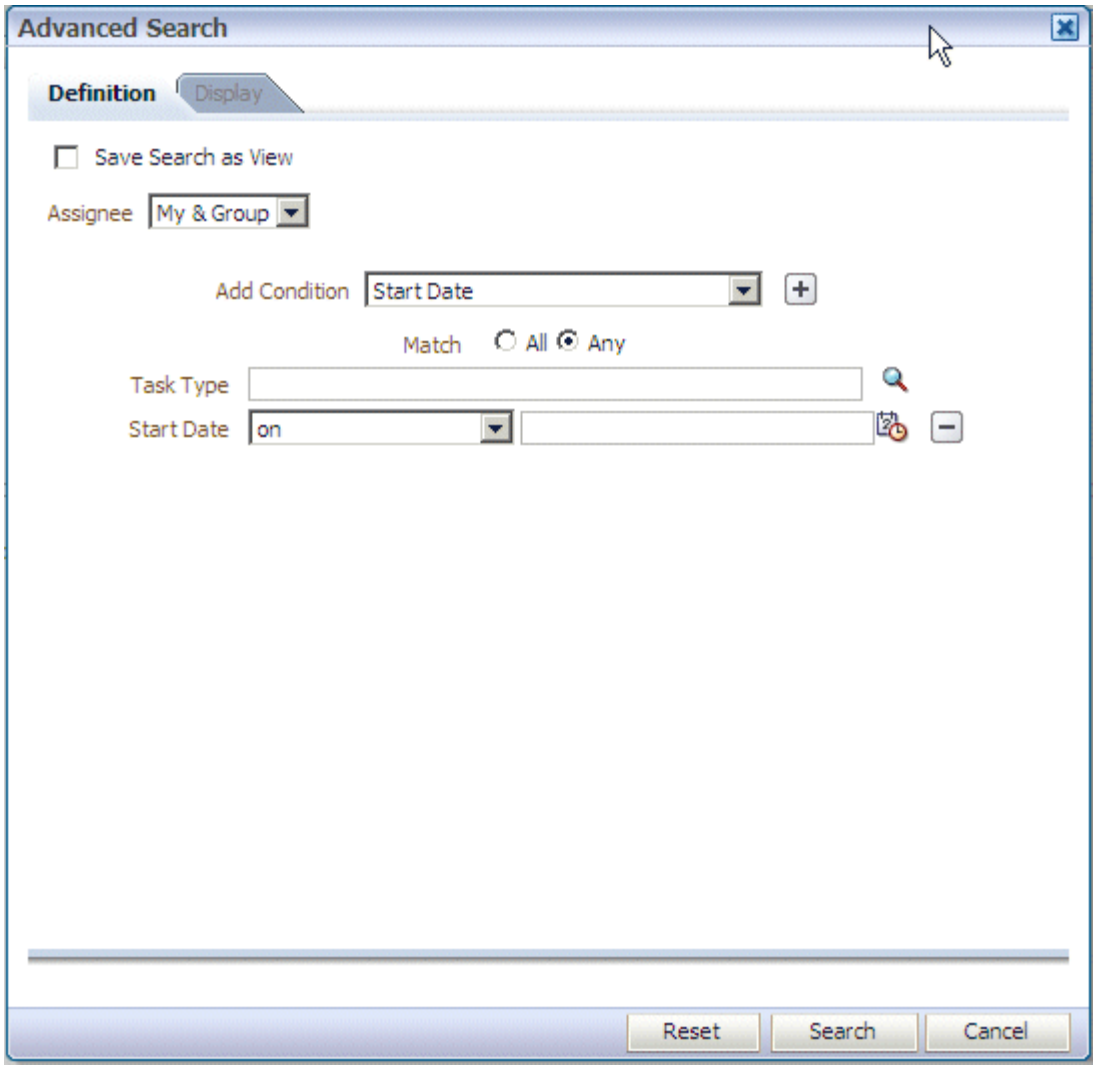
Condition	Description
Start Date	The start date of the task (used with ToDo tasks).
Assignees	The current task assignees (users, groups or application roles).
Task Definition Name	The name of the task component that defines the task instance.
Owner Role	The application role (if any) that owns the task 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



6. Select **Any** or **All** for matching multiple filters.
7. (Optional) Search on a task type.
8. Click **Search**.

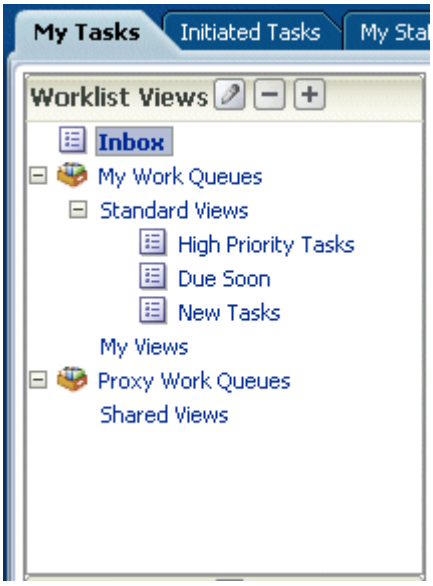
   The task list page with the tasks filtered according to your criteria appears.

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

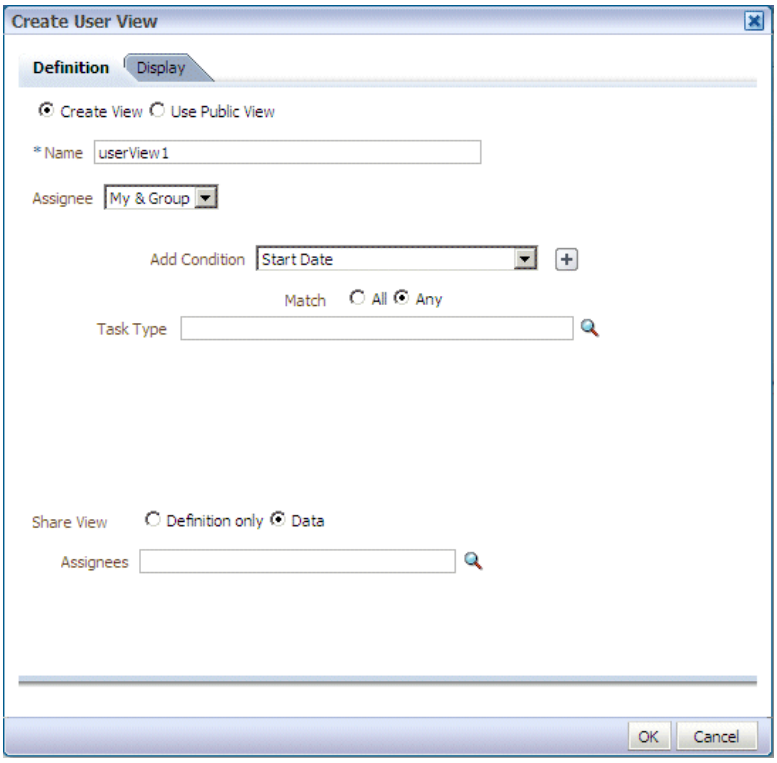


Use **Worklist Views** to create, share, and customize views.

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.

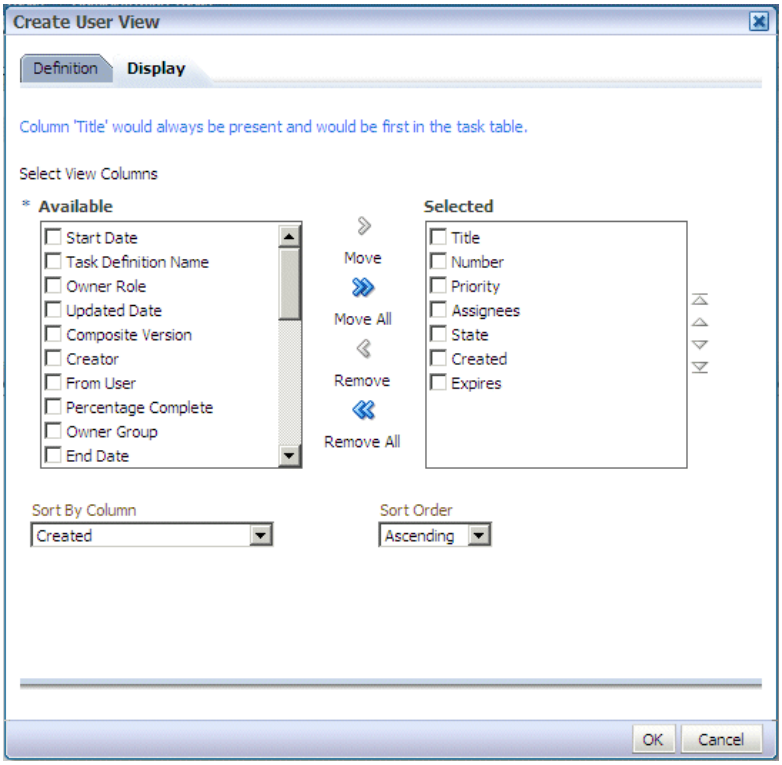
Figure 29–11 Creating a Worklist View



- **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

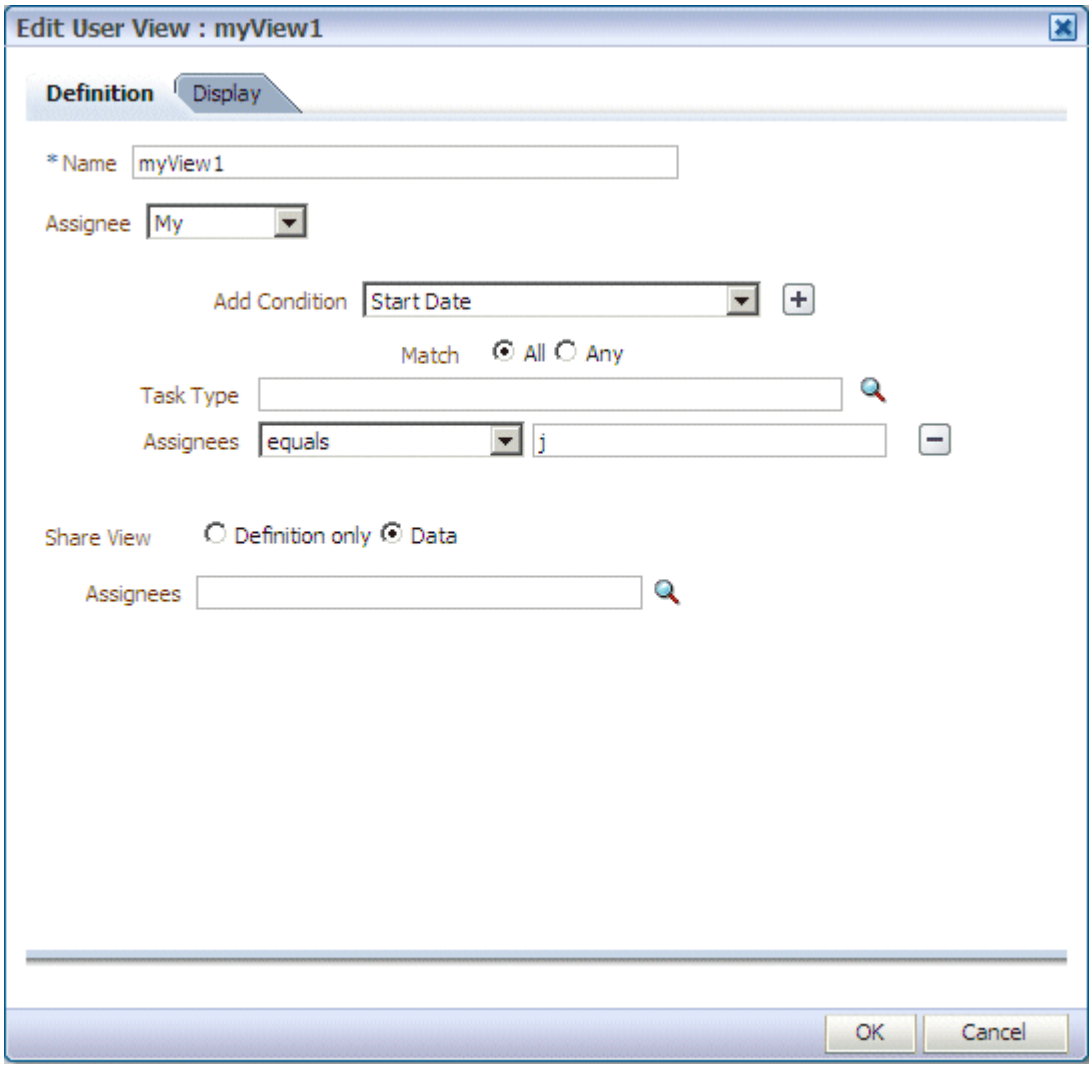
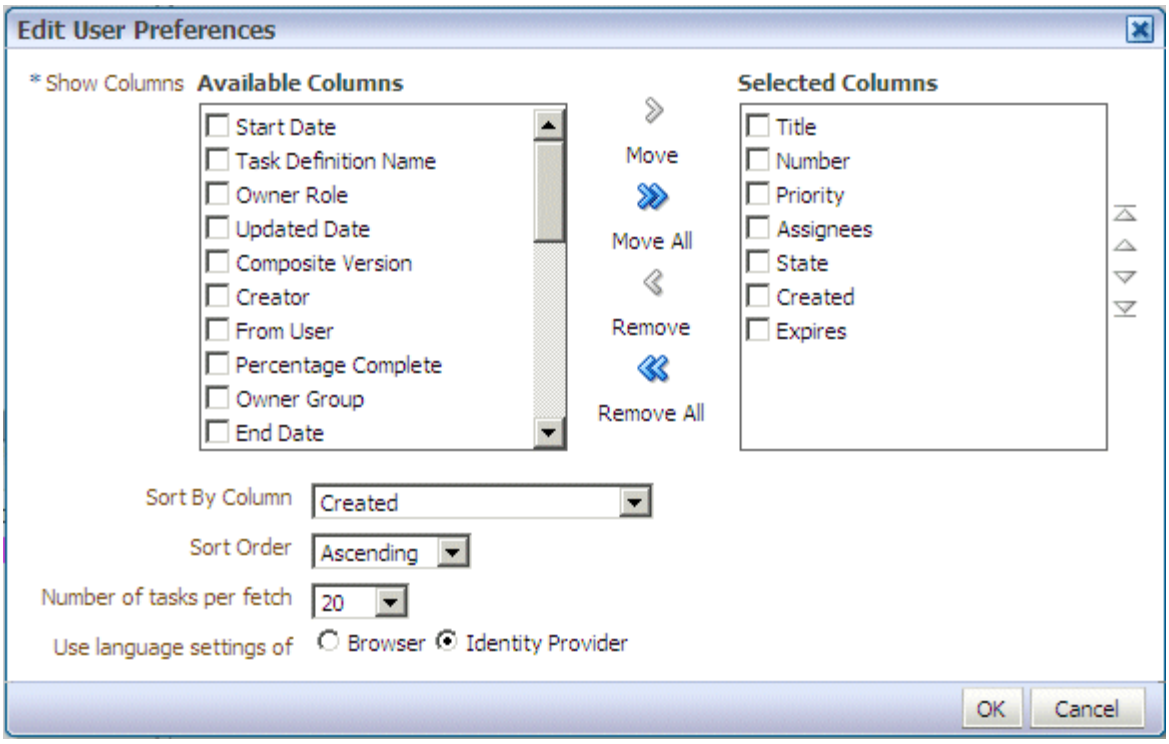


Figure 29–14 Customizing Fields in a Worklist View



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:

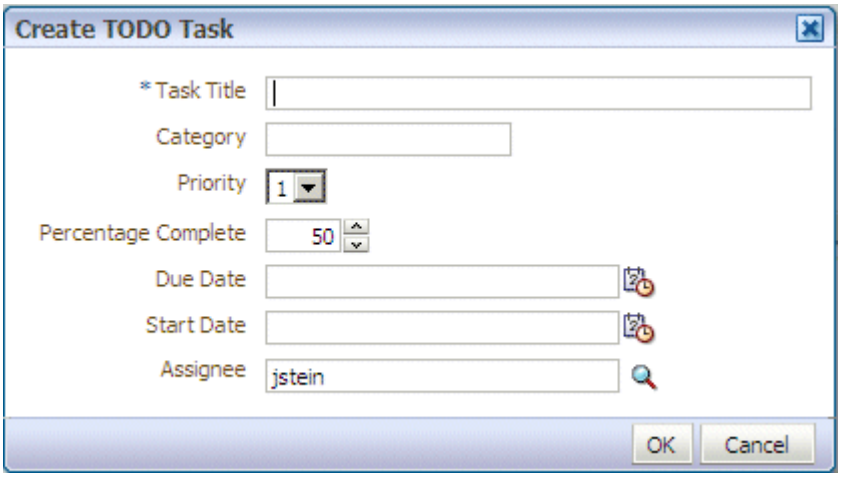
1. Click the **Edit** icon.
2. Add or remove status states for display, as shown in Figure 29–15, and click **OK**.

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.

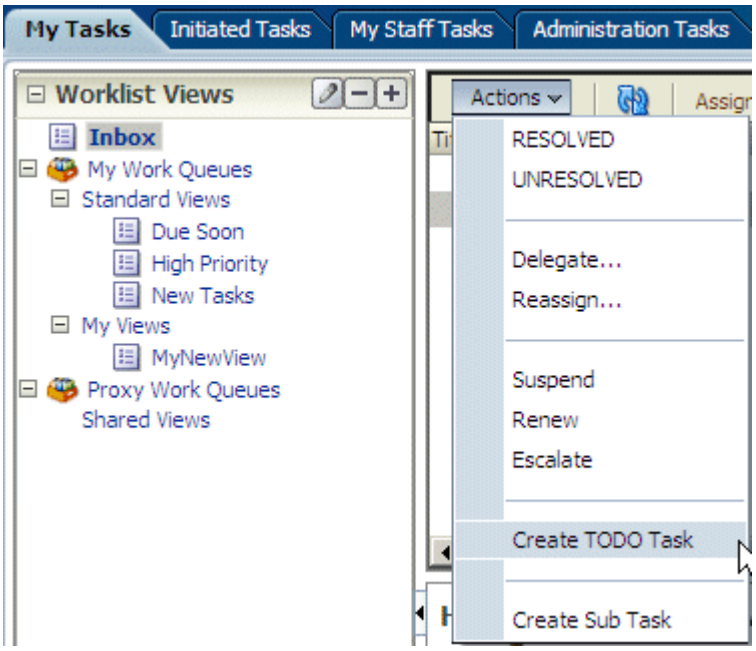
ToDo tasks can be reassigned, escalated, and so on, and deleted (logical delete) and purged (physical delete). Reassignment, escalation, and so on of the parent task does not affect the assignment of any child ToDo tasks. The completion percentage of a ToDo task can be reset to less than 100% after it 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

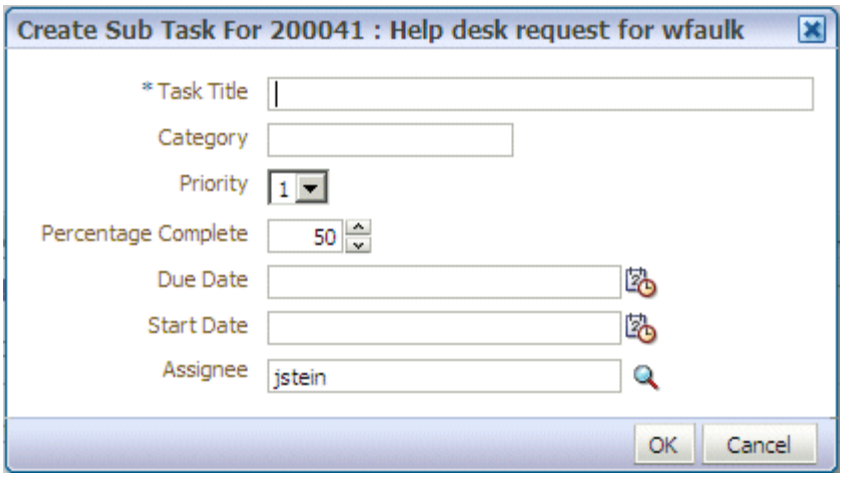


2. Provide details in the Create ToDo Task dialog, shown in Figure 29–17, and click **OK**.
   - **Task Title**: Enter anything that is meaningful to you.
   - **Category**: Enter anything that is meaningful to you.
   - **Priority**: Select from **1** (highest) to **5** (lowest)
   - **Percentage Complete**: 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



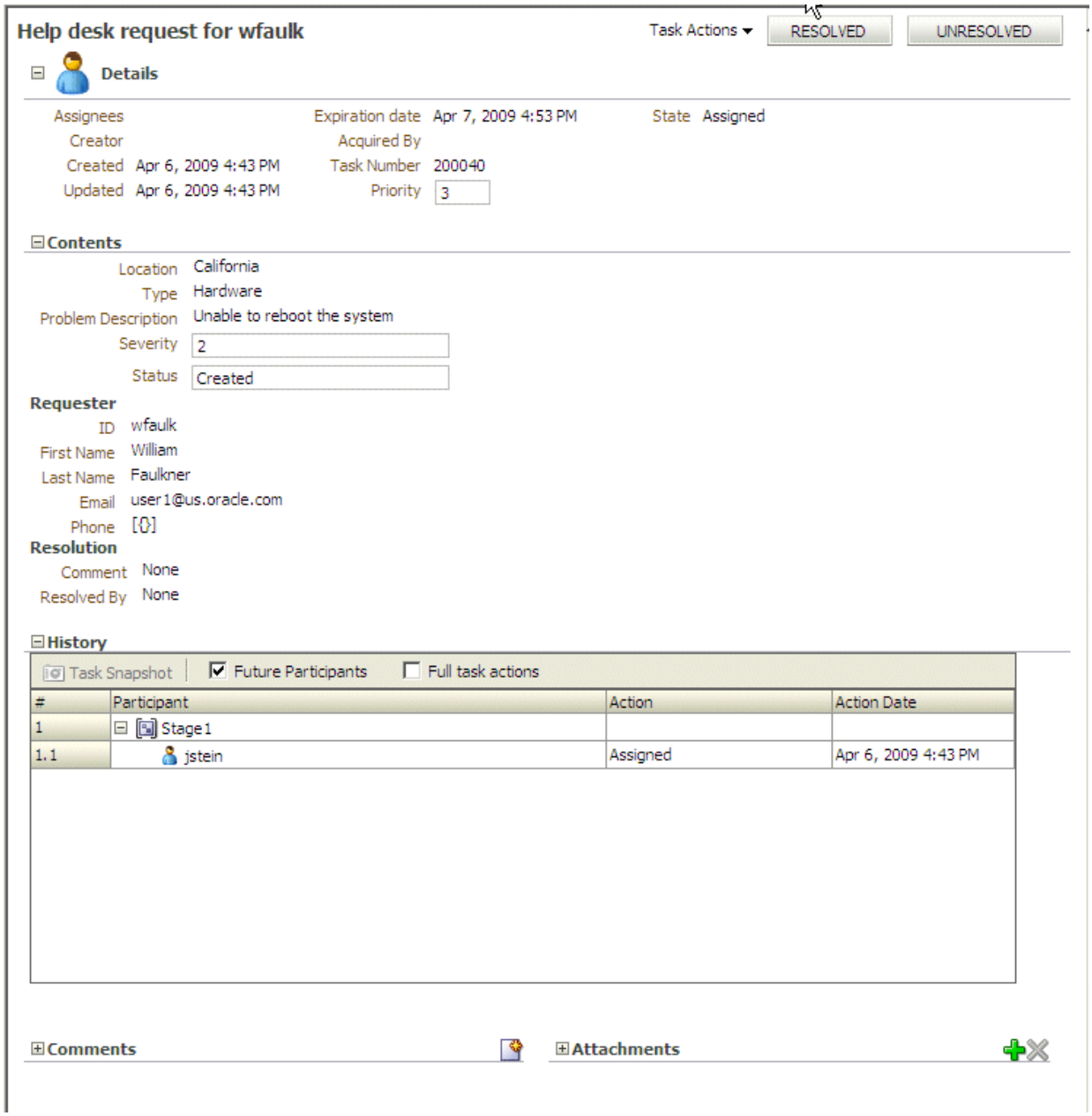
Subtasks can break down a business task into measurable subtasks, and can be created for ToDo tasks also. Multiple levels of subtasks are not supported (that is, you cannot have subtasks inside of subtasks). If you create multiple levels of subtasks, and attempt to act on the main task (for example, 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



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 the these states, the task, task history, and subtasks (parent task in parallel flow) can be viewed. If a task is suspended, then it can only be resumed or withdrawn. A task that is assigned to a group must be claimed before any actions can be performed on it.

Note: If you act on a task from the task details page, for example, if you approve a task, then any unchanged task details data is saved along with the saved changes to the task. However if you act on a task from the Actions menu, then unchanged task details are not saved.

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

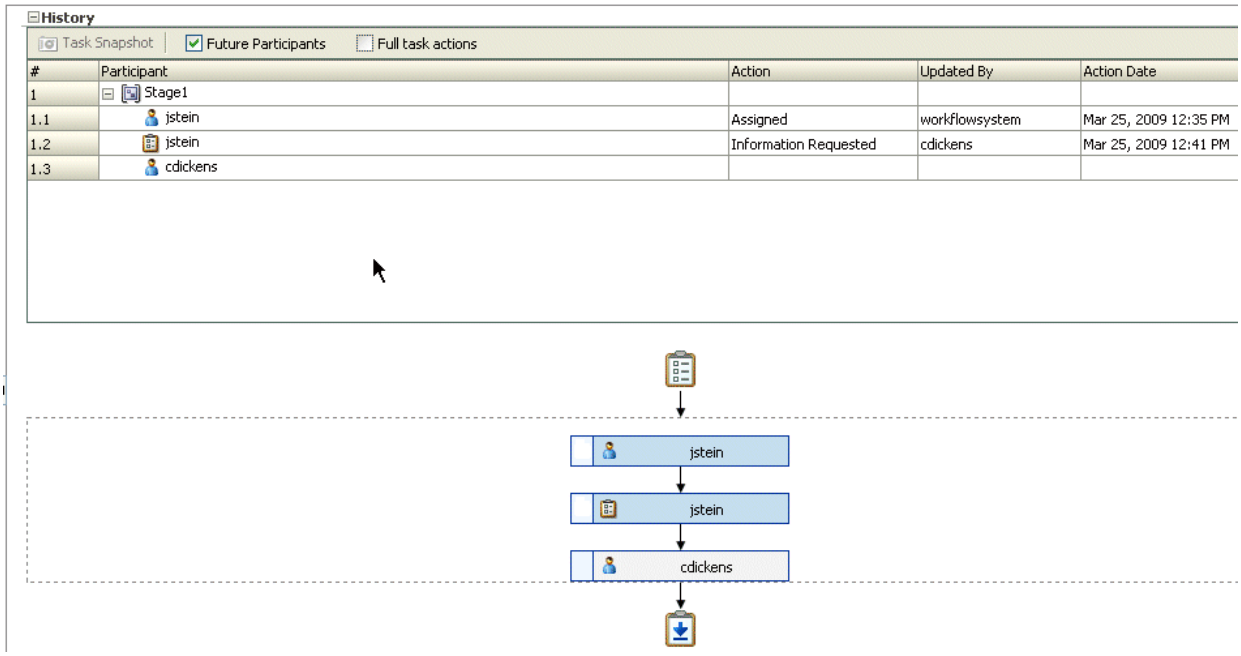
You can include the following actions in the short history list by modifying the `shortHistoryActions` element.

- 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

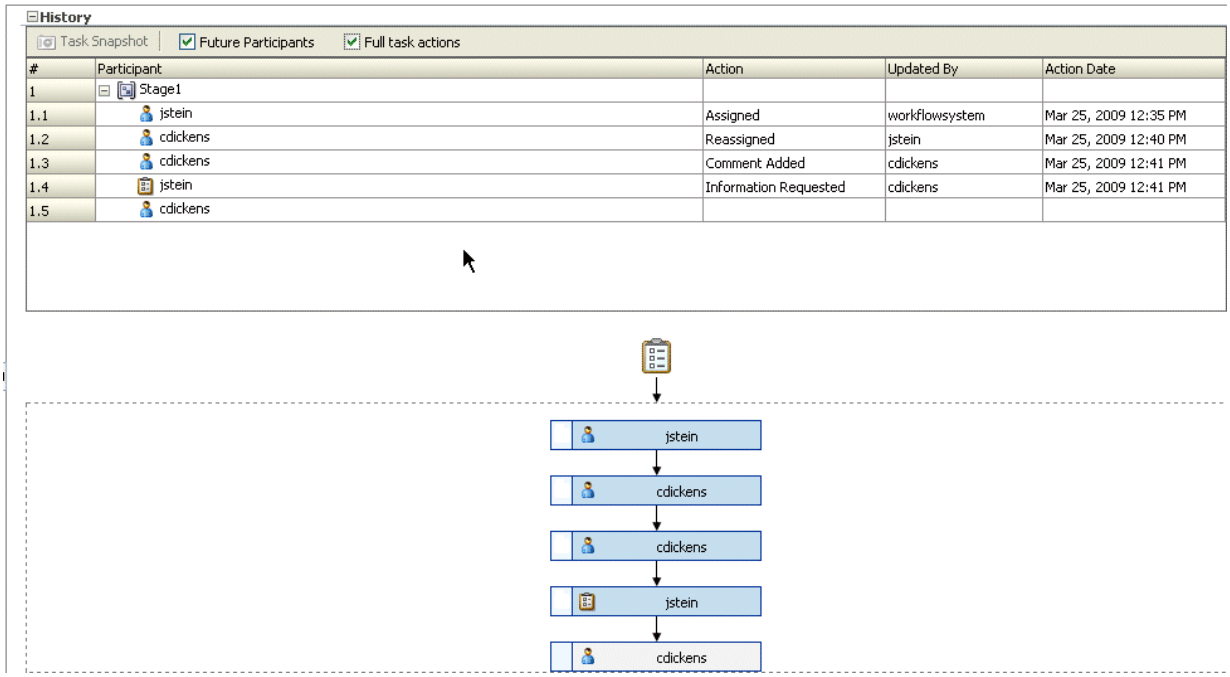
The history provides a graphical view of a task flow, as shown in Figure 29–20.

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



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

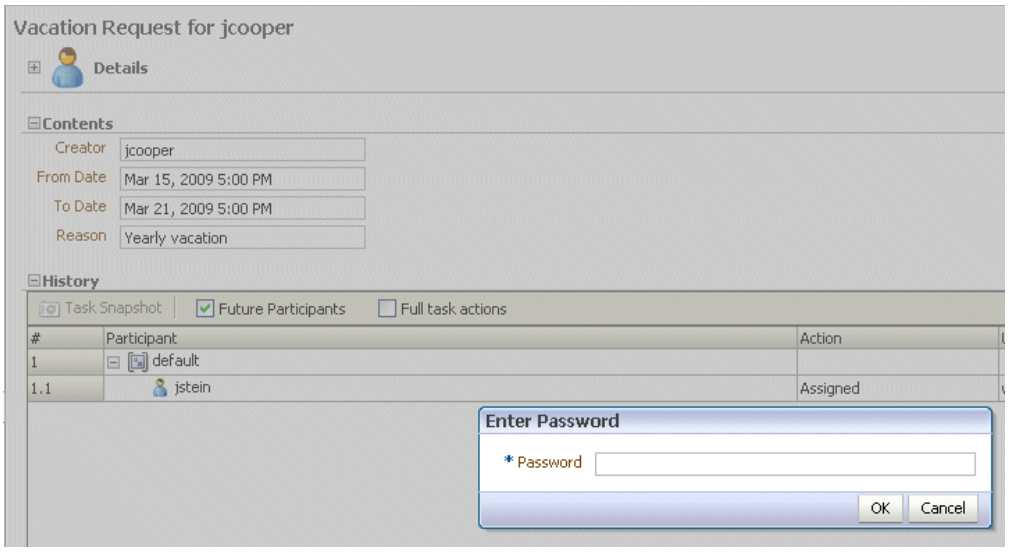
If the human task was designed to permit ad hoc routing, or if no predetermined sequence of approvers was defined, then the task can be routed in an ad hoc fashion in the worklist. For such tasks, a **Route** button appears on the task details page. From the Route page, you can look up one or more users 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

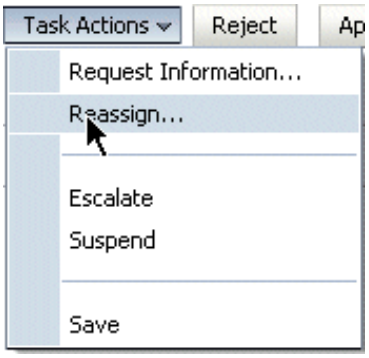


Note: Any kind of change to the task details page, such as changing a priority or adding a comment, requires you to save the change. If you add an attachment to a task, it is automatically saved.

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

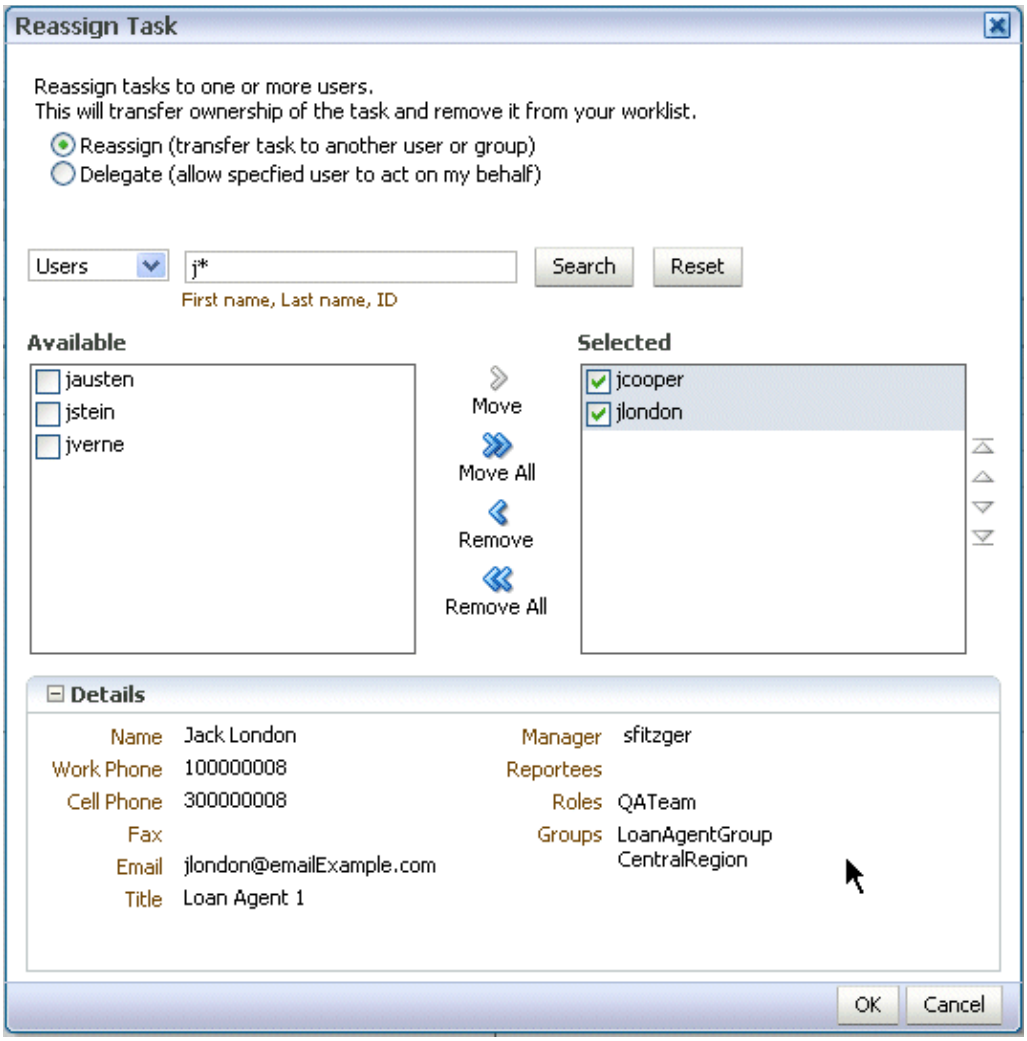


2. Select **Reassign** or **Delegate**.

   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

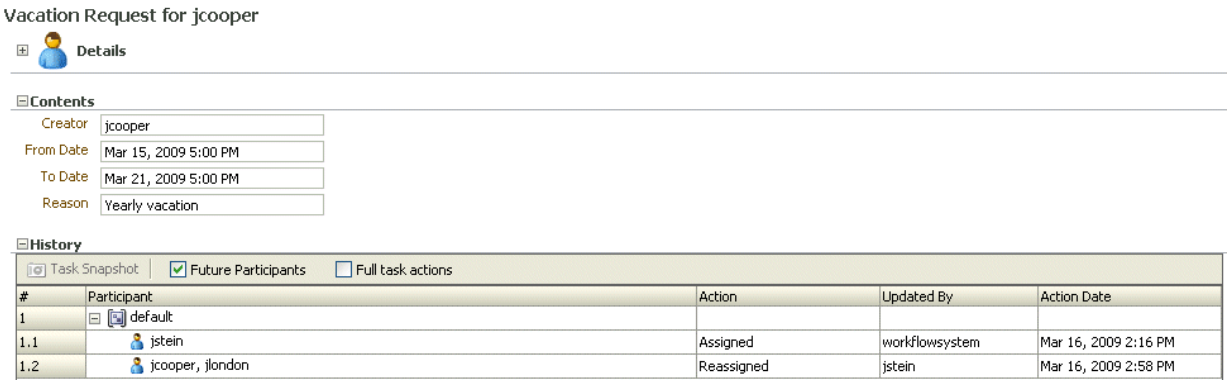


A supervisor can always reassign tasks to any of his reportees.

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.

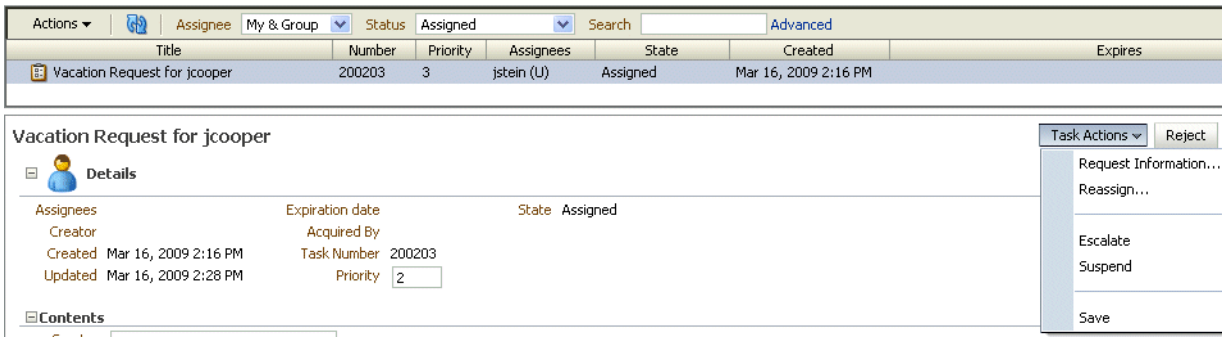
Figure 29–25 Claiming a Task



To request information:

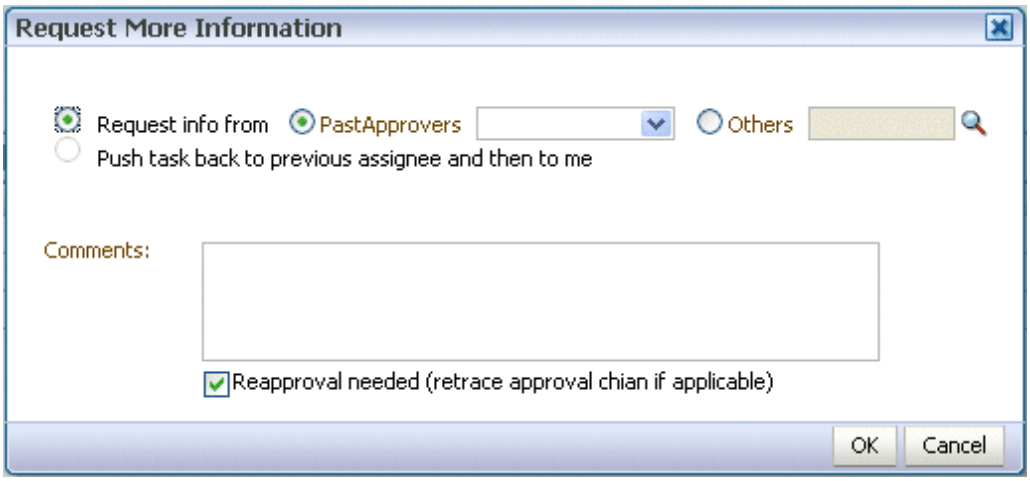
1. From the **Task Actions** list, select **Request Information**, as shown in Figure 29–26.

Figure 29–26 Requesting Information



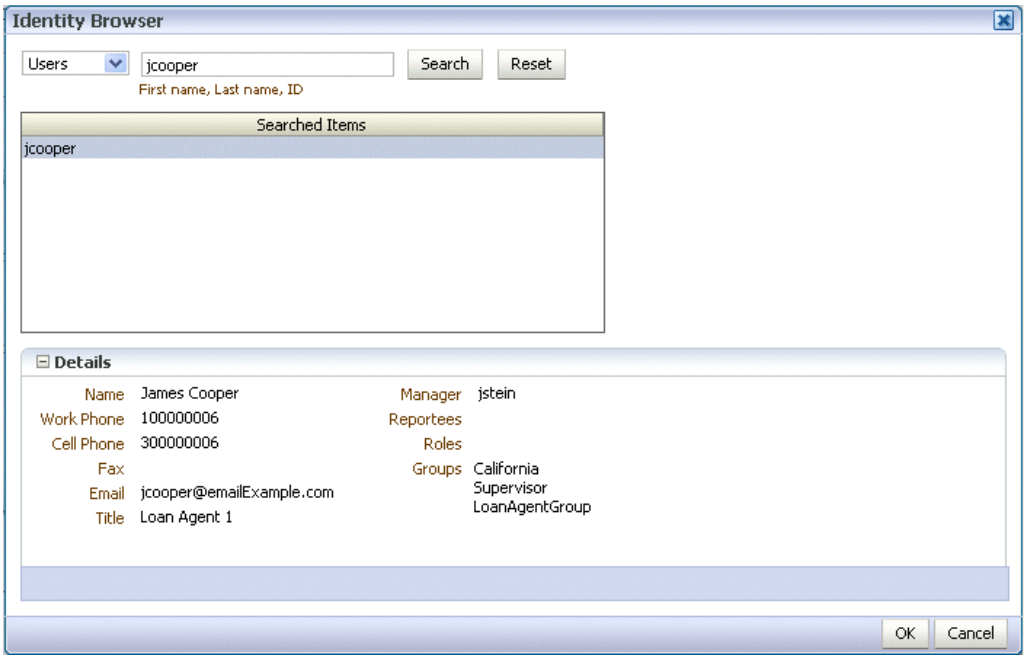
2. Request information from a past approver or search for a user name, or push the task back to the previous assignee, as shown in Figure 29–27.

Figure 29–27 Requesting Information from Past Approvers or Another User, or Pushing the Task Back



If you use the **Search** icon to find a user name, the Identity Browser appears, as shown in Figure 29–28.

Figure 29–28 Identity Browser



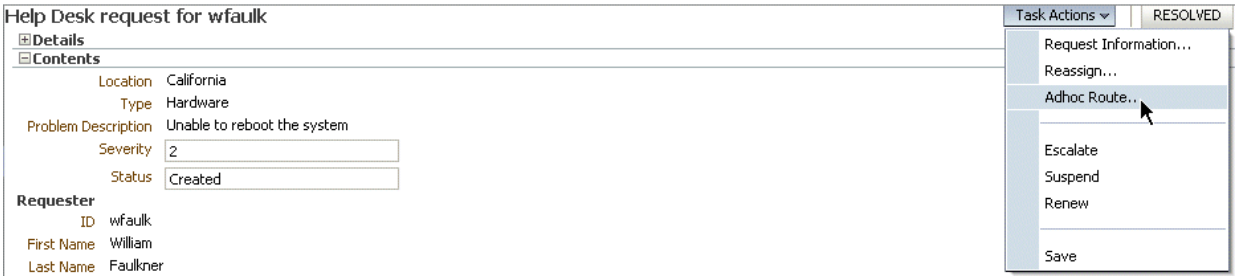
Note: If you are in a multi-tenancy environment, search for a user simply by the user identifier and not by the tenant identifier. For example, if the user identifier is jstein and the tenant identifier is company_name.jstein, you would search by using jstein.

3. Click **OK**.

To route a task:

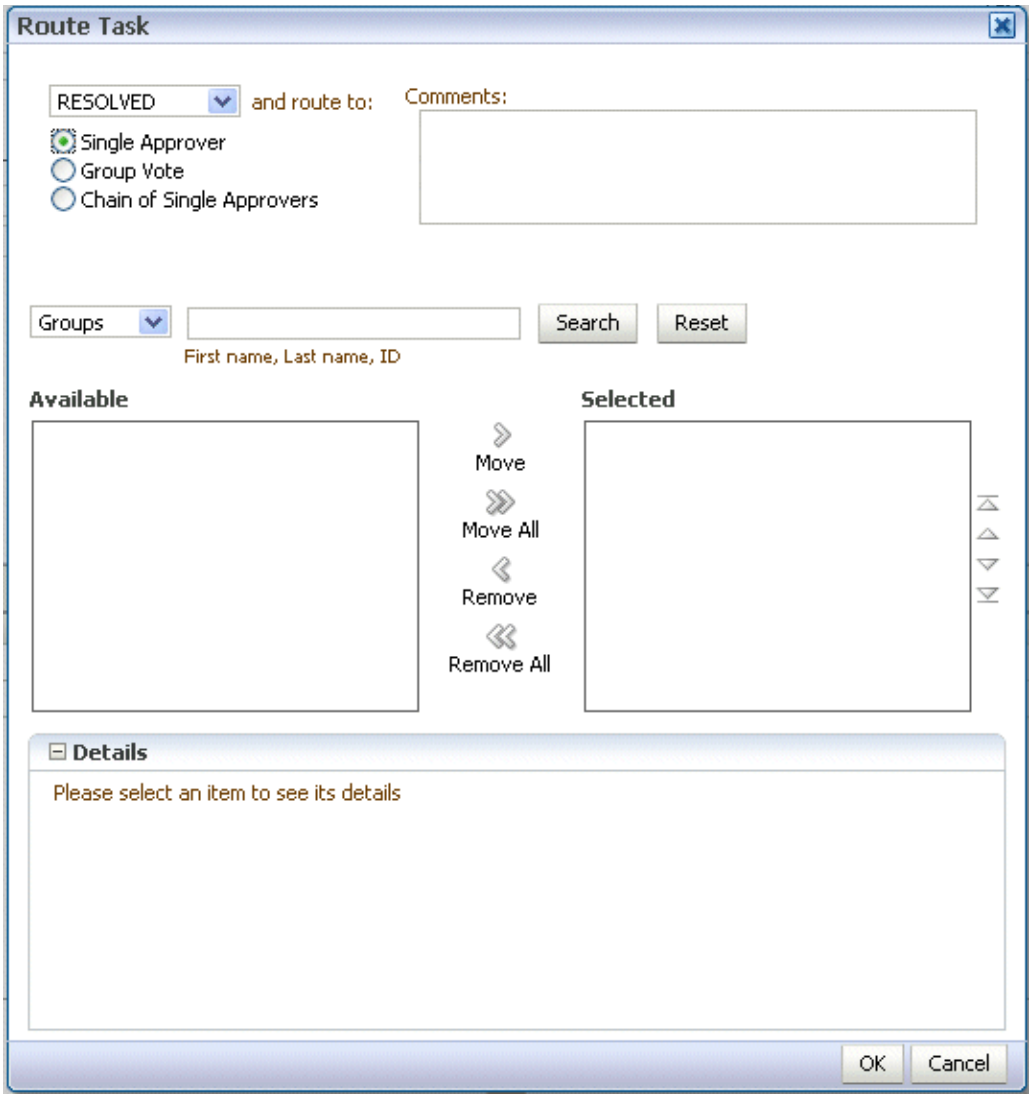
1. From the **Task Actions** list, select **Adhoc Route**, as shown in Figure 29–29.

Figure 29–29 Ad Hoc Routing



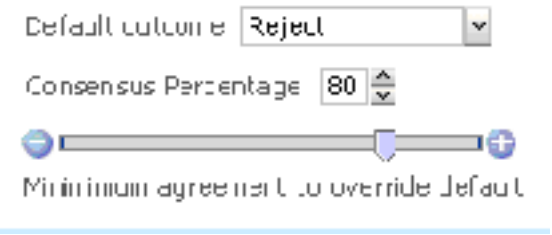
2. Select an action and a routing option, as shown in Figure 29–30.

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



- **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**.

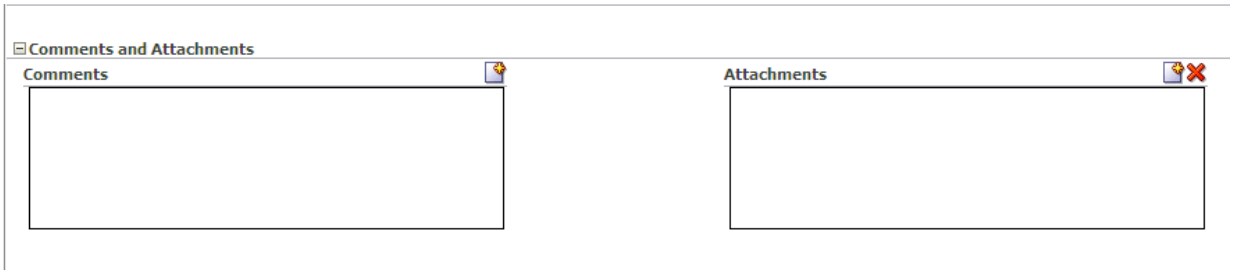
To add comments or attachments:

Notes:

- Click **Save** before you browse for or upload attachments, 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**.

1. In the **Comments** or **Attachments** area, click **Add**.

Figure 29–32 Worklist Comments and Attachments

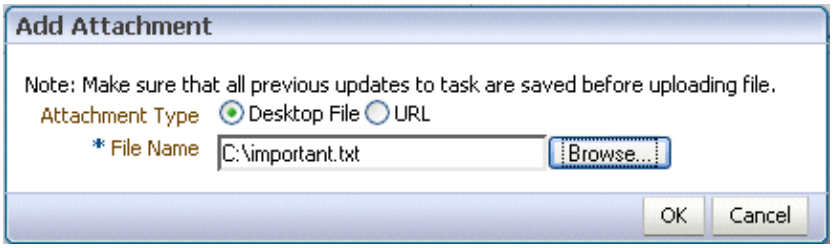


2. Enter comment text and click **OK**. Note that comments cannot be deleted once they are added.

   The date and timestamp and your user name are included with the comment.

3. For attachments, provide a file or URL attachment, as shown in Figure 29–33, and click **OK**.

Figure 29–33 Adding a Worklist Attachment



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.

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*.

---

4. From the Task Actions list, click **Save**.

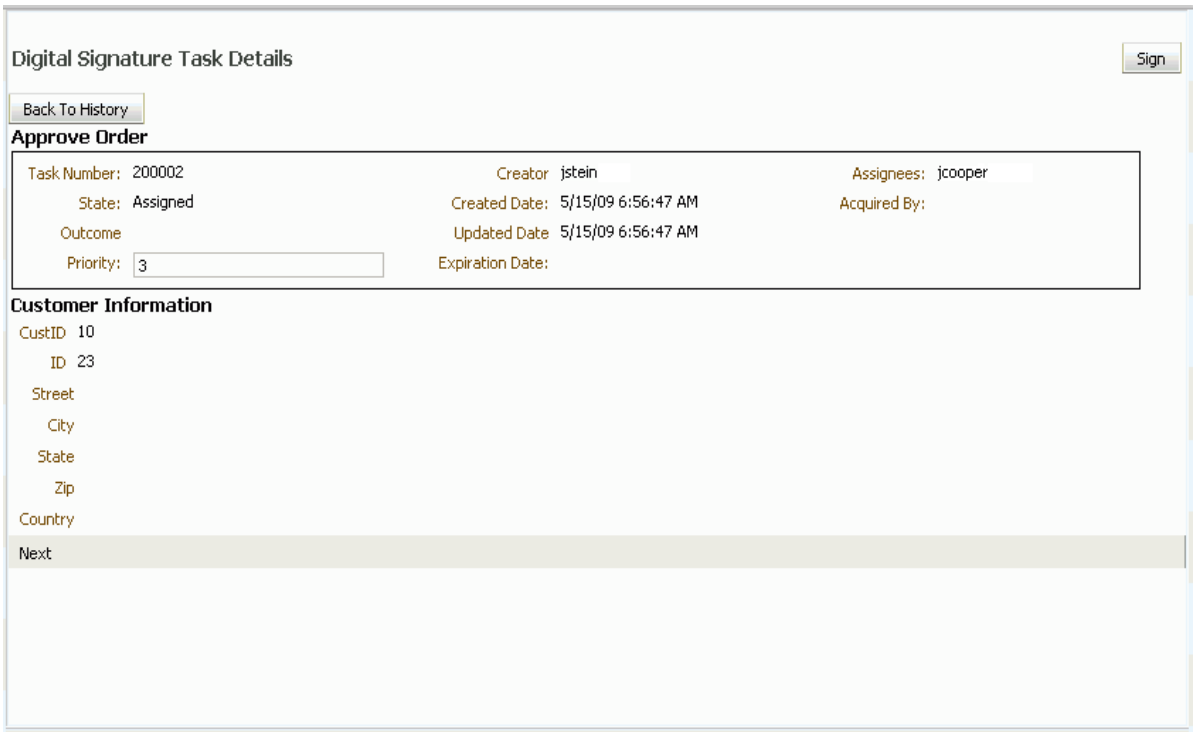
29.4.4 How To Act on Tasks That Require a Digital Signature

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.

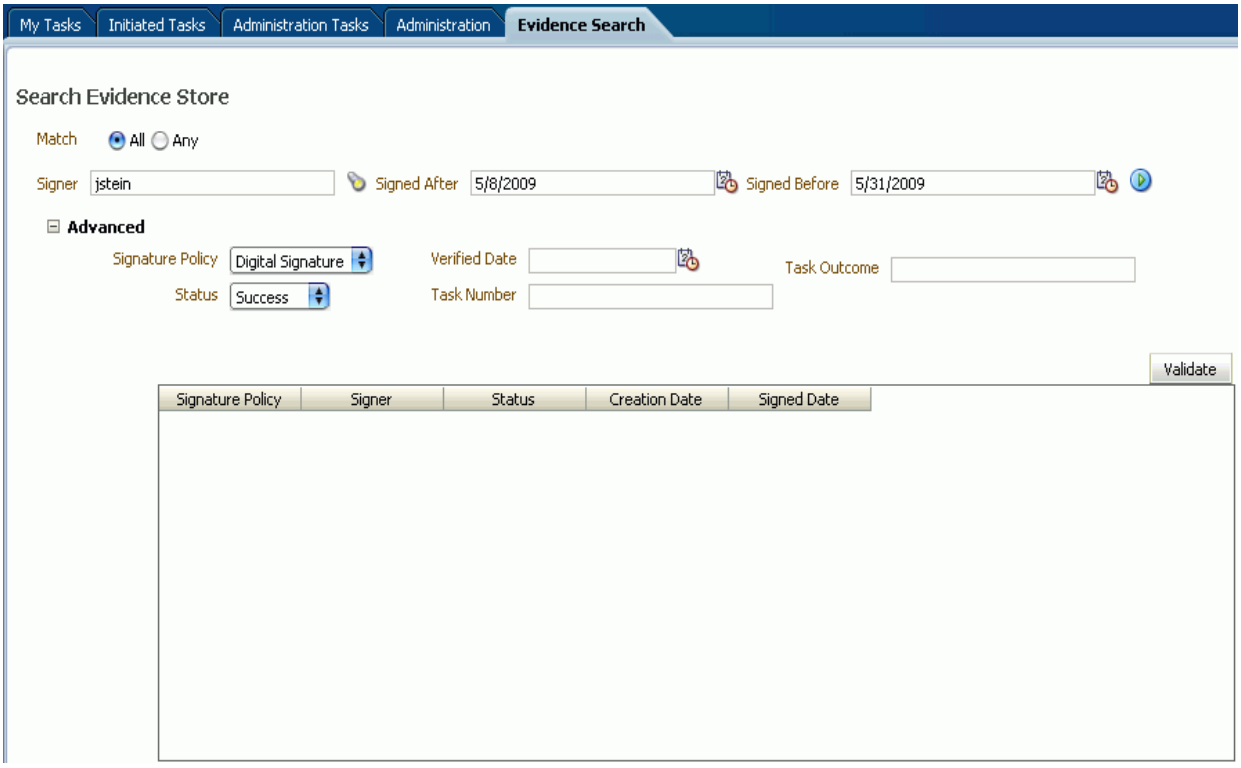
When you act on a task that has a signature policy, the **Sign** button appears, as shown in Figure 29–34.

Figure 29–34 Digital Signature Task Details



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



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:

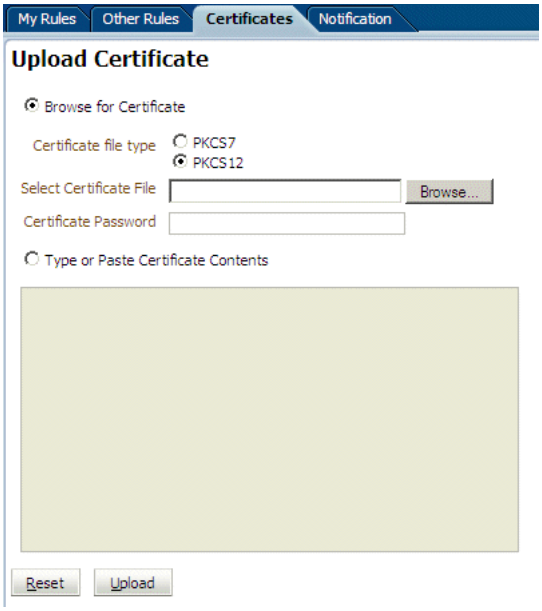
   a. Export the chain from Mozilla Firefox as a .P12 file (PKCS12 format that also contains your private key).

   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

Table 29–8 (Cont.) Task Actions and Approvers

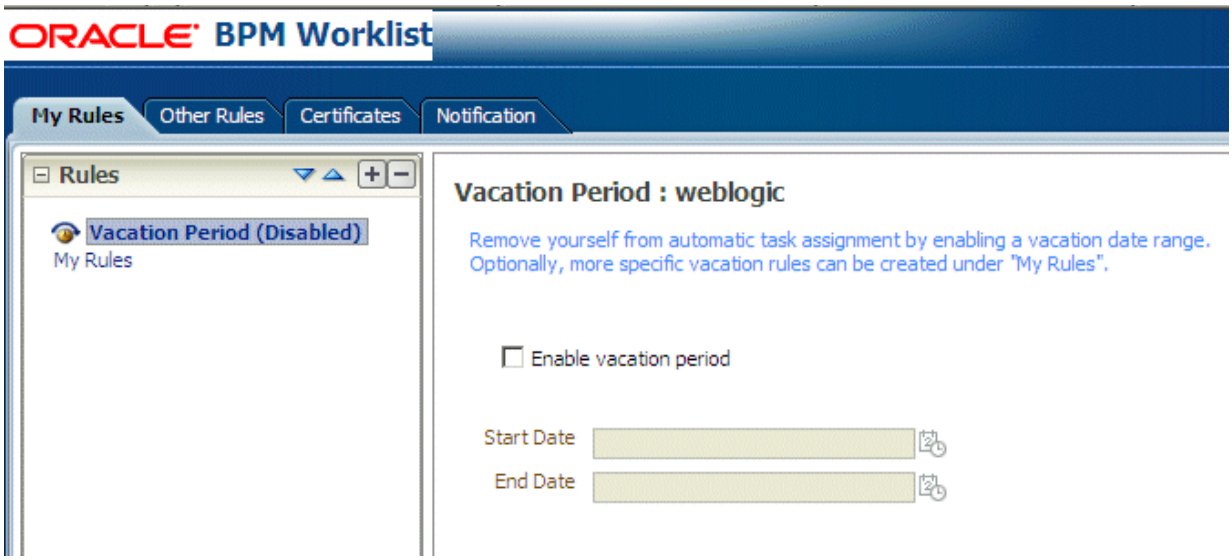
Task Action	Admin	Owner (+ Owner Group)	Assignee (+ Assignee Manager + Assignee Group + Proxy Assignee)	Creator	Reviewer	Approver
Reassign	Yes[5]	Yes[5]	Yes (No for proxy assignee)	No	No	No
Release	Yes	Yes	Yes	No	No	No
Renew	No	Yes	Yes	No	No	No
Resume	Yes	Yes	Yes	No	No	No
Route	No	Yes	Yes	No	No	No
Skip 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
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.

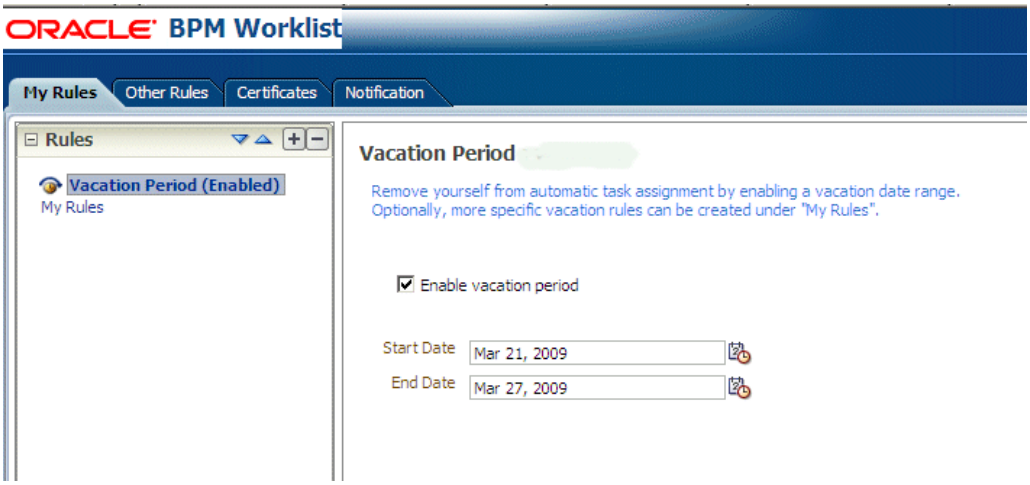
2. Click **Enable vacation period**.

3. Provide start and end dates.

4. Click **Save**.

   The vacation period is enabled, as shown in Figure 29–38.

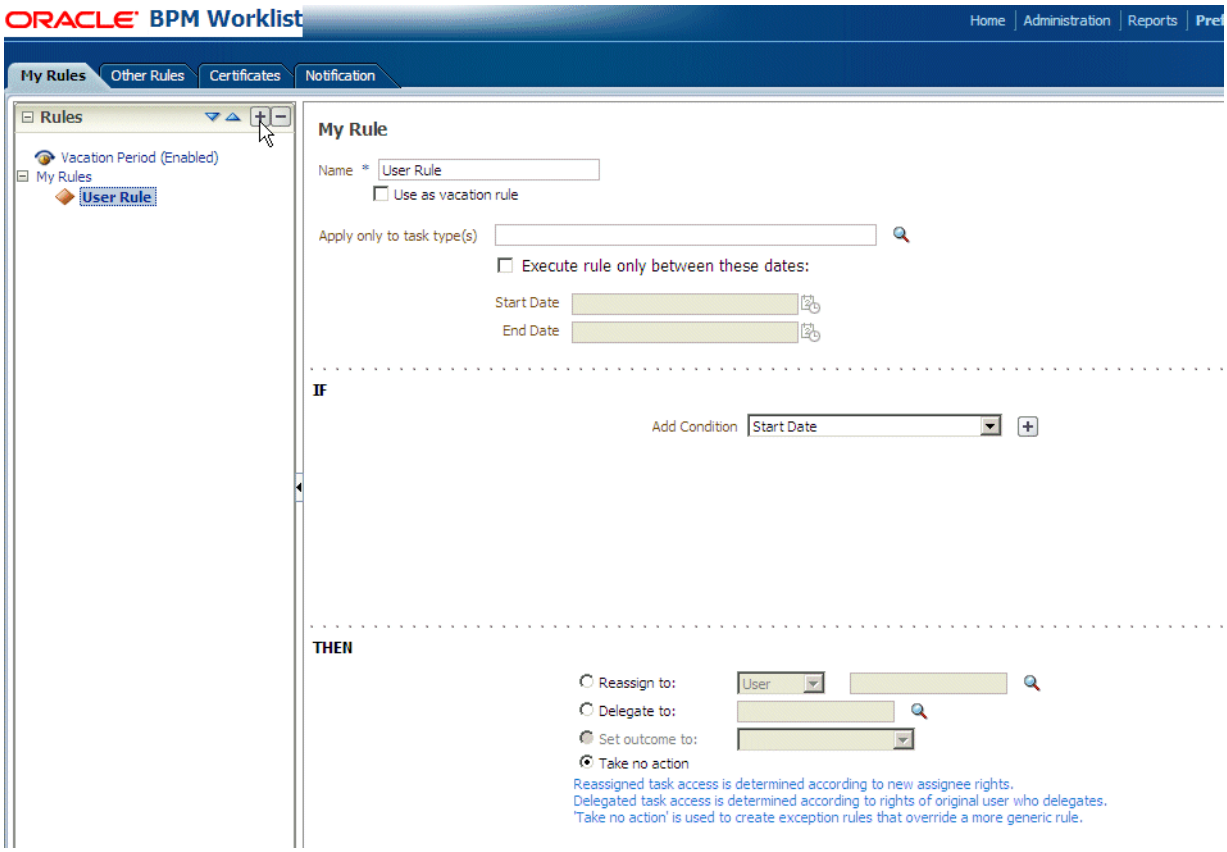
Figure 29–38 Enabling a Vacation Period



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



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 **Preference**s 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:

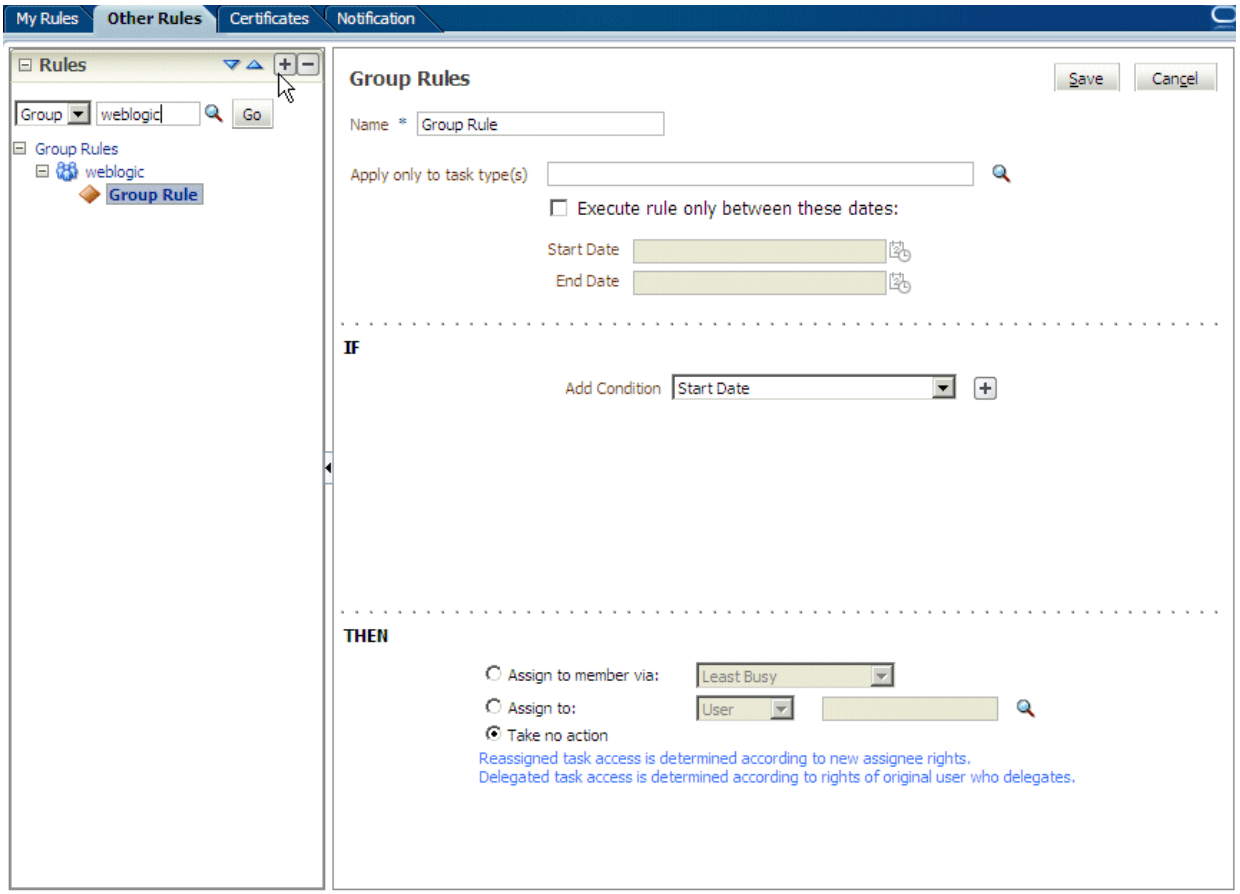
- **Assign to member via**—You can specify a criterion to determine which member of the group gets the assignment. This dynamic assignment criterion can include round-robin assignment, assignment to the least busy group member, or assignment to the most productive group member. You can also add your 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 **Preference**s 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)
- Mapping 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.

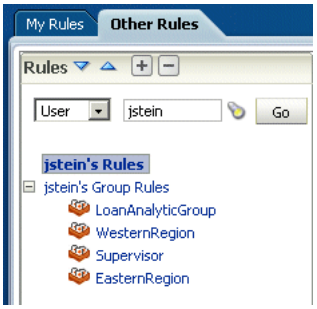
29.8.1 How To Manage Other Users' or Groups' Rules (as an Administrator)

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:

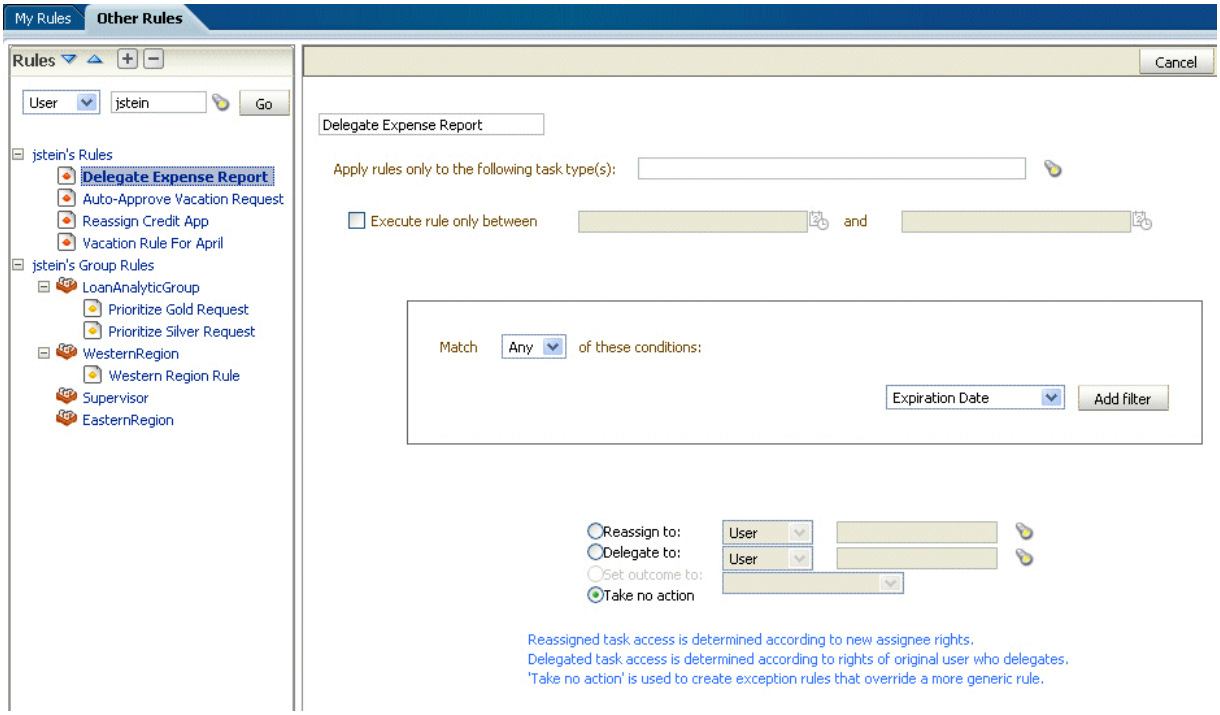
1. From the task list page, click the **Rules** link.
2. Click the **Other Rules** tab.
3. Search for the user or group for whom rules are to be created, as shown in Figure 29–41.

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 Set the Worklist Display (Application Preferences)

Application preferences customize the appearance of the worklist. Administrators can specify the following:

- **Login page realm label**—If the identity service is configured with multiple realms, then the Oracle BPM Worklist login page displays a list of realm names. `LABEL_LOGIN_REALM` specifies the resource bundle key used to look up the label to display these realms. The term *realm* can be changed to fit the user community—terms such as *country*, *company*, *division*, or *department* may be more appropriate. Administrators can customize the resource bundle, specify a resource key for this string, and then set this parameter to point to the resource key.
- **Global branding icon**—This is the image displayed in the top left corner of every page of the worklist. (The Oracle logo is the default.) Administrators can provide a `.gif`, `.png`, or `.jpg` file for the logo. This file must be in the `public_html` directory.
- **Resource bundle**—An application resource bundle provides the strings displayed in the worklist. By default, this is the class at:

  `oracle.bpel.worklistapp.resource.WorklistResourceBundle`

  Administrators can change the strings shown in the application by copying `WorkflowResourceBundle` and creating their own. This parameter allows administrators to specify the class path to this custom resource bundle.

  Administrators must extend `WorklistResourceBundle.java` by adding their resource strings. Administrators can change the strings shown in the application by copying `WorkflowResourceBundle` and creating their own. This parameter allows administrators to specify the class path to this custom resource bundle. Then administrators create a JAR file from the compiled resource bundle and copy it under

```
SOA_Oracle_Home\j2ee\home\applications\worklist\worklist\WEB-INF\lib
```

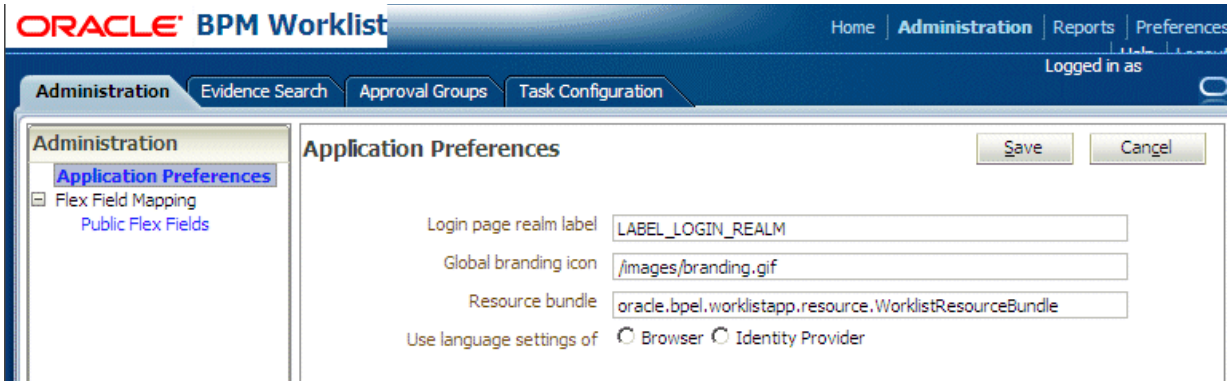
- **Use language settings of**—Select the browser or the identity provider.

  The Identity Provider that stores information on worklist users can store the user's locale, which can determine the worklist display language. Alternatively, the user's browser can supply the locale information. This parameter determines which is used as the source for determining the worklist application display language.

To specify application preferences:

1. Click the **Administration** tab.
2. Click **Application Preferences**.
3. Browse for the locations of the application preferences (login page realm label, branding icon, or resource bundle), as shown in Figure 29–43.

Figure 29–43 Application Preferences



4. Select which language settings you want to use—from the browser or the identity provider.
5. Click **Save**.

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
Source	String
Amount	Number
Role	String
Priority	String

Table 29–10 (Cont.) Predefined Attributes for Messaging Filters

Attribute	Data Type
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.

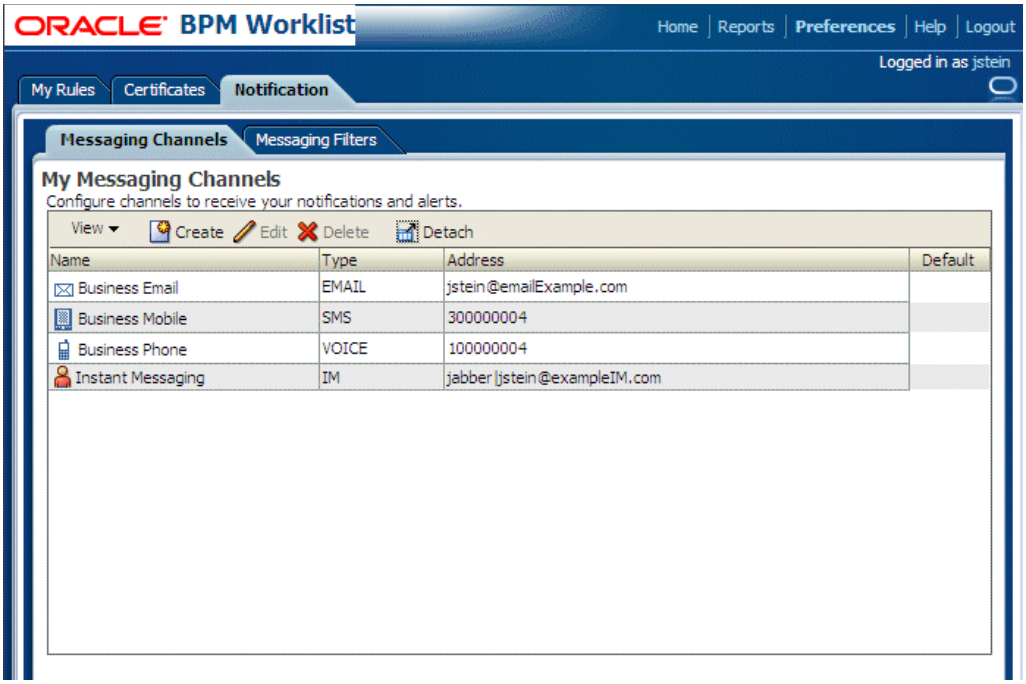
To view messaging channels:

1. Click the **Preferences** link.
2. Click the **Notification** tab.
3. Click the **Messaging Channels** tab.

   The My Messaging Channels list appears (Figure 29–44) 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 29–44 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.

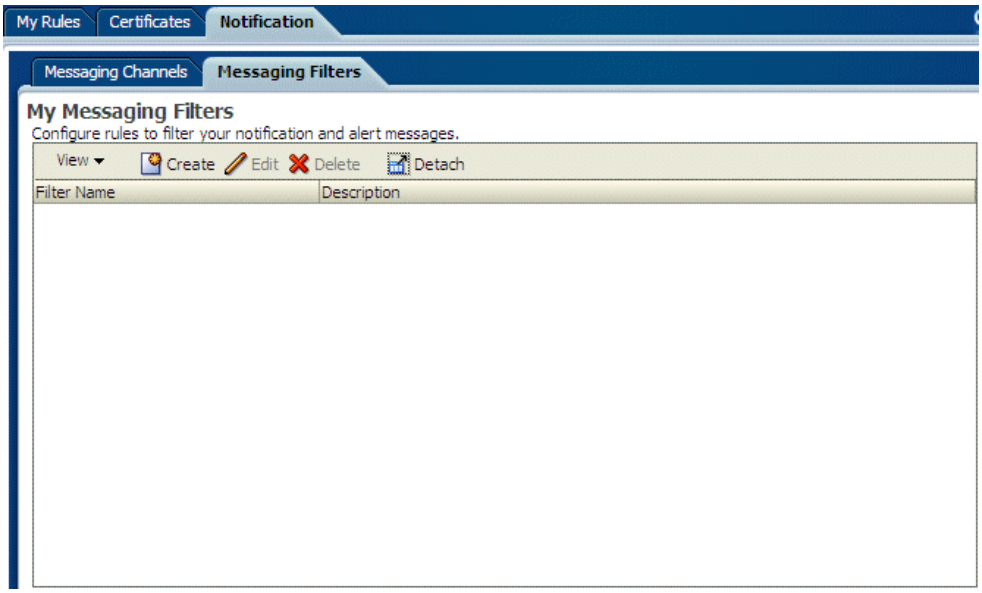
To view your messaging filters:

1. Click the **Notification** tab.
2. Click the **Messaging Filters** tab.

   The My Messaging Filters list appears (Figure 29–45) and displays the following information:

   - **Filter Name**: The name of the messaging filter
   - **Description**: An optional description of the messaging filter

Figure 29–45 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.

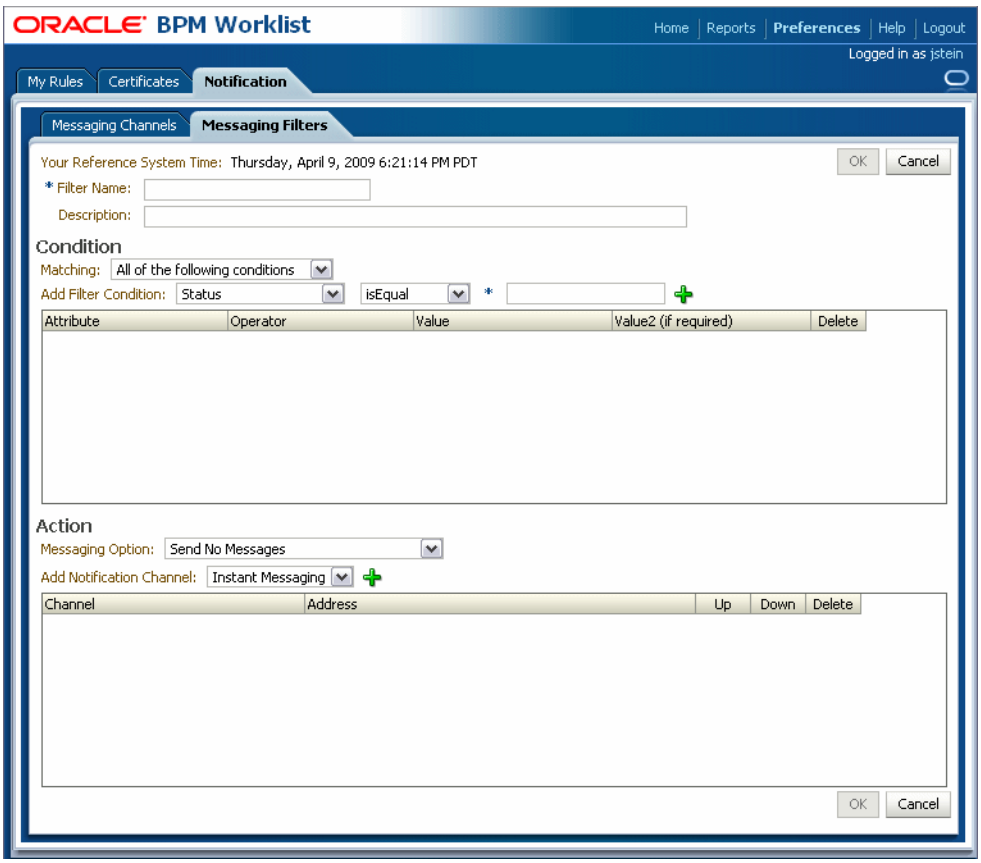
29.9.4.2 Creating Messaging Filters

To create a messaging filter:

1. Click **Create**.

   The Messaging Filters page appears, as shown in Figure 29–46.

Figure 29–46 Adding a Messaging Filter



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:

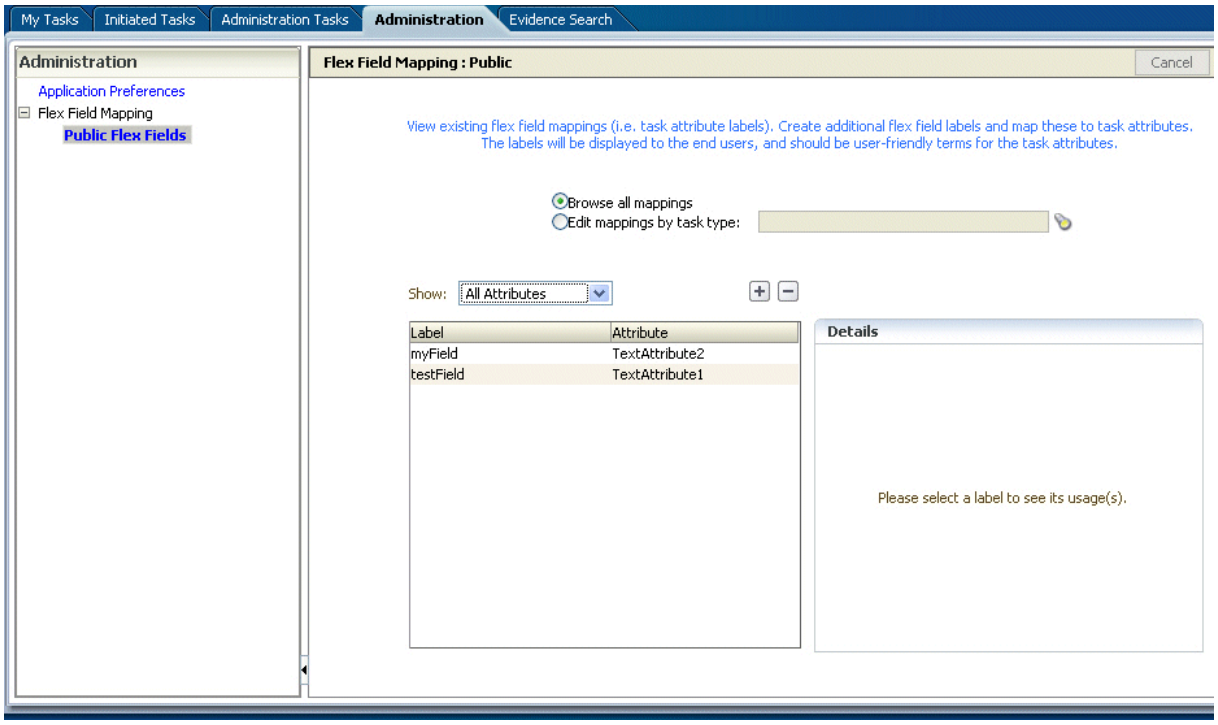
- Mapped attributes must be defined before instances of the business process are generated. Only instances generated after mapped attributes are created reflect the correct mapped attributes. Older instances of the business process do not reflect subsequent mapped attribute changes.
- When you add a 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 29–47, 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.

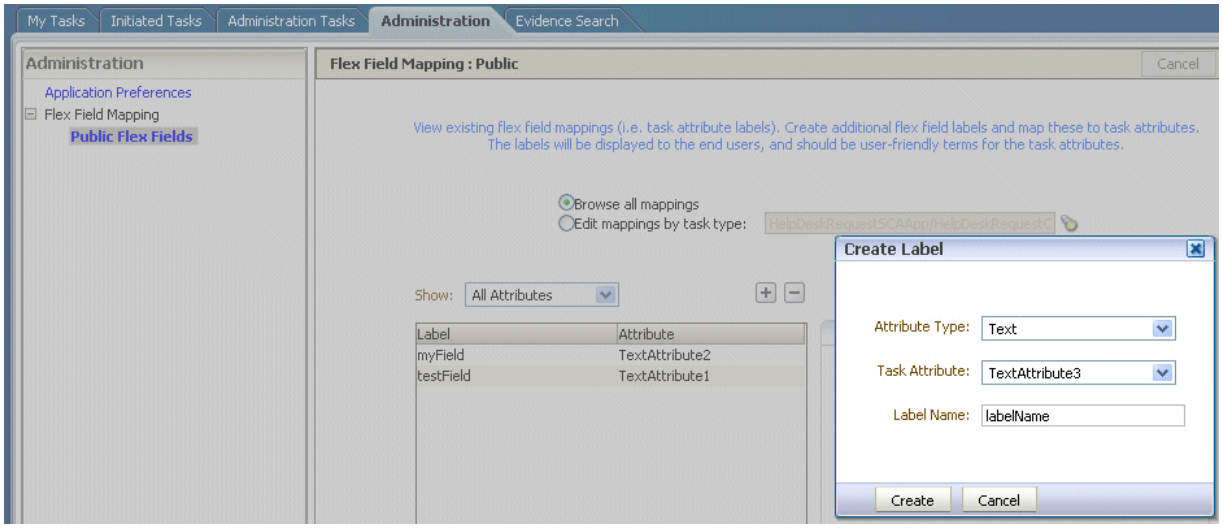
Figure 29–47 Mapped Attribute Mapping



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 29–48.

Figure 29–48 Creating a Label



As Figure 29–48 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.

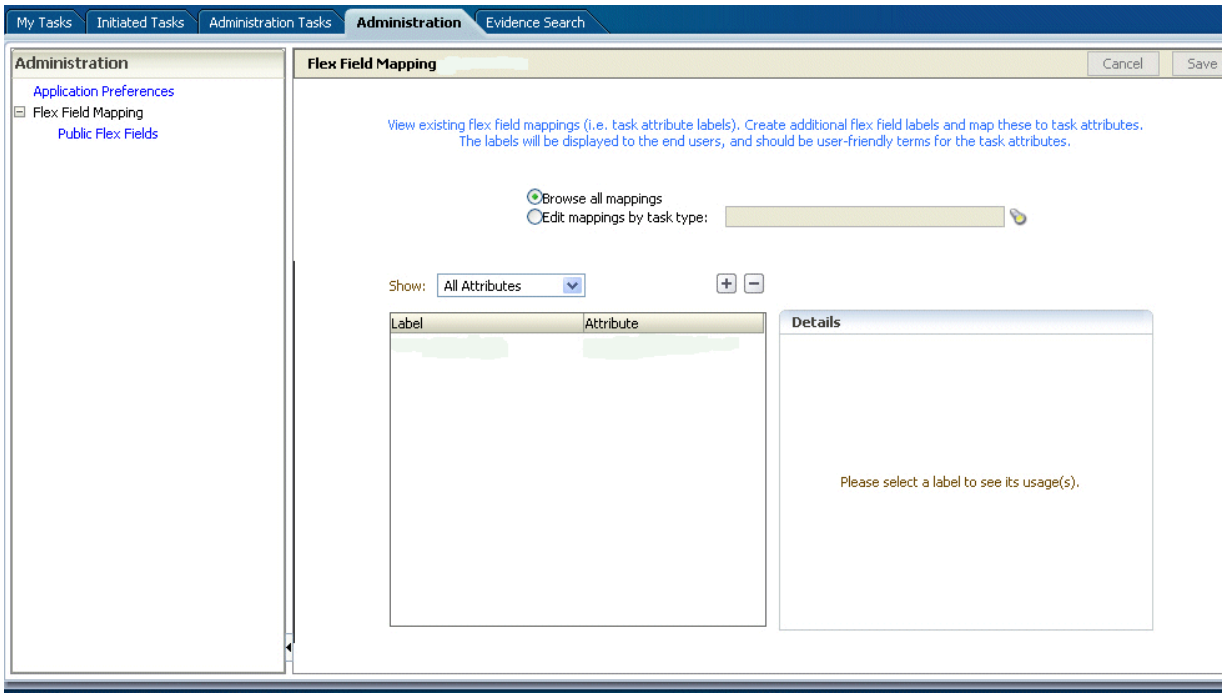
Note the following restrictions:

- 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.

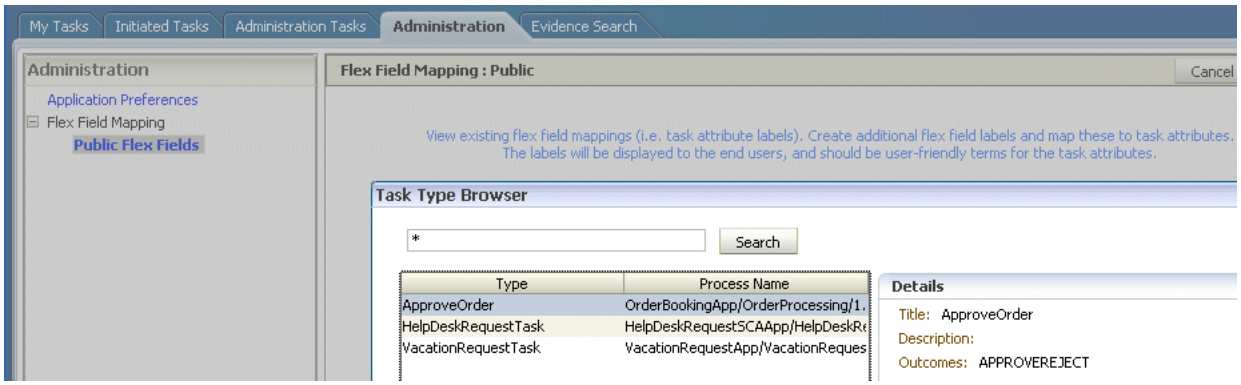
Figure 29–49 Browsing Mappings



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**.

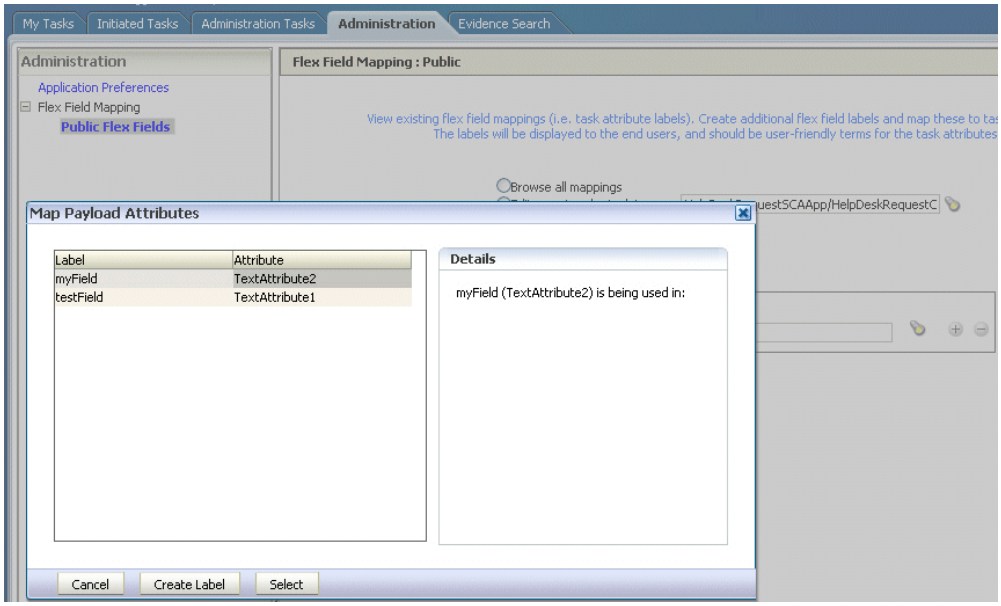
Figure 29–50 Selecting a Task Type



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 29–51.

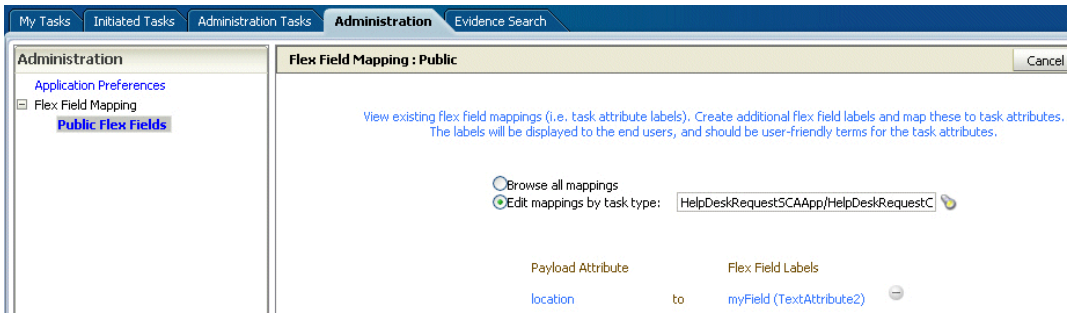
Figure 29–51 Selecting a Label



4. Select a mapping label and click **Select**.

   Figure 29–52 shows a completed mapping.

Figure 29–52 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
customerAttributeString1	String
customerAttributeString2	String
customerAttributeNumber1	Double
customerAttributeNumber2	Double
customerAttributeDate1	Date
customerAttributeDate2	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 by 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 by 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 by 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.

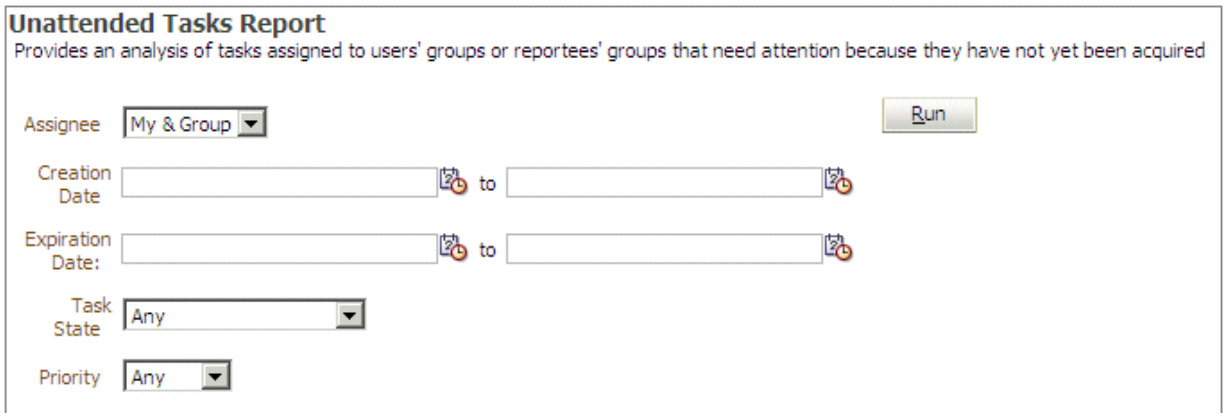
   Figure 29–53 shows the report types available.

Figure 29–53 Oracle BPM Worklist Reports

3. Provide inputs to define the search parameters of the report.

   Figure 29–54 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 29–54 Unattended Tasks Report—Input Page for Task Analysis

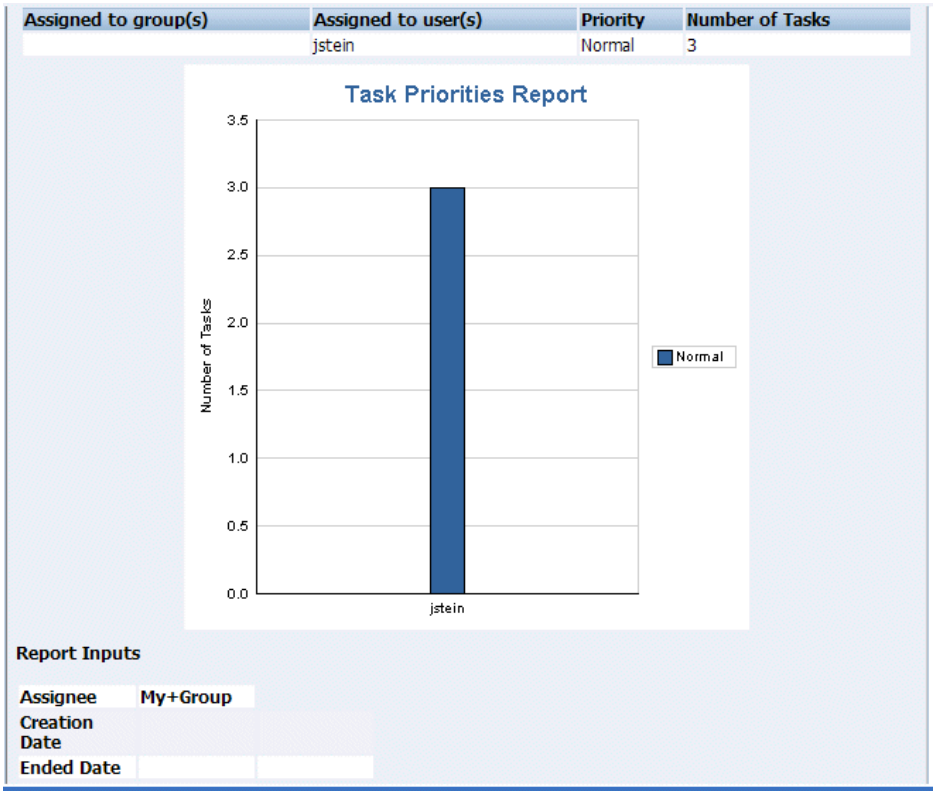


4. Click **Run**.

29.11.2 What Happens When You Create Reports

As shown in Figure 29–55, 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).

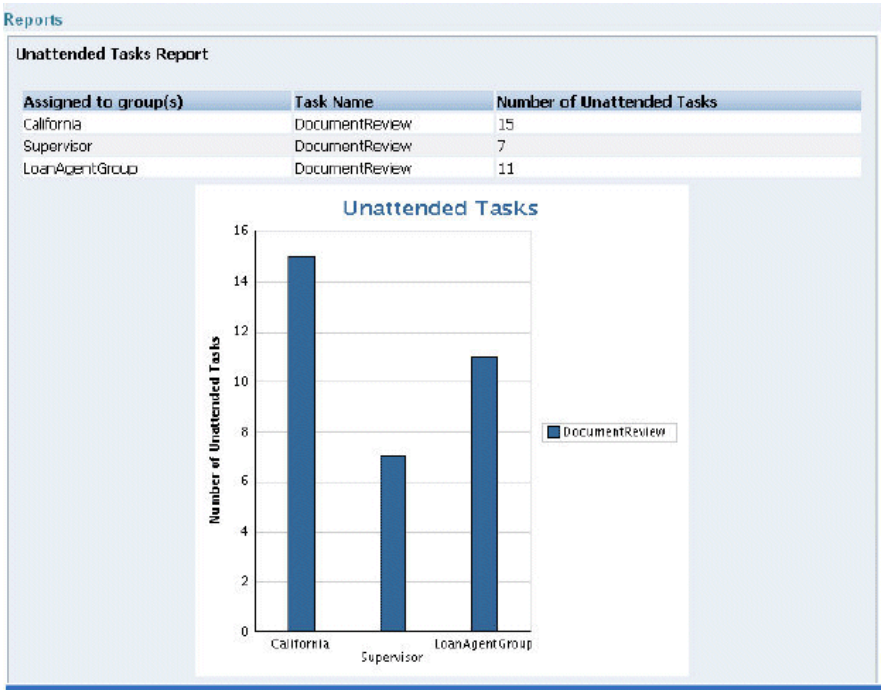
Figure 29–55 Report Display—Table Format, Bar Chart Format, and Report Inputs



29.11.2.1 Unattended Tasks Report

Figure 29–56 shows an example of an Unattended Tasks report.

Figure 29–56 Unattended Tasks Report



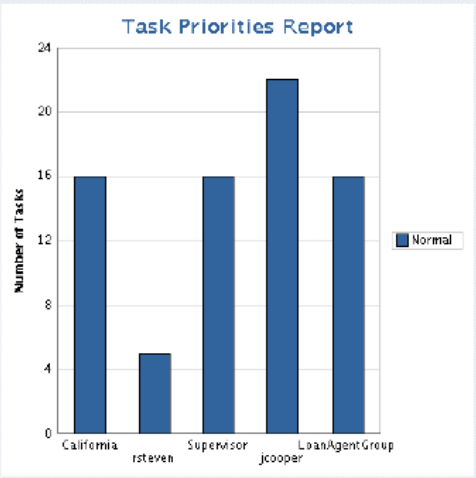
The report shows that the California group has 15 unattended tasks, the Supervisor group has 7 unattended tasks, and the LoanAgentGroup has 11 unattended tasks. The unattended (unclaimed) tasks in this report are all DocumentReview tasks. If multiple types of unattended task exists when a report is run, 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 29–57 shows an example of a Tasks Priority report.

Figure 29–57 Tasks Priority Report

Assigned to group(s)	Assigned to user(s)	Priority	Number of Tasks
California		Normal	16
	rsteven	Normal	5
Supervisor		Normal	16
	jcooper	Normal	22
LoanAgentGroup		Normal	16

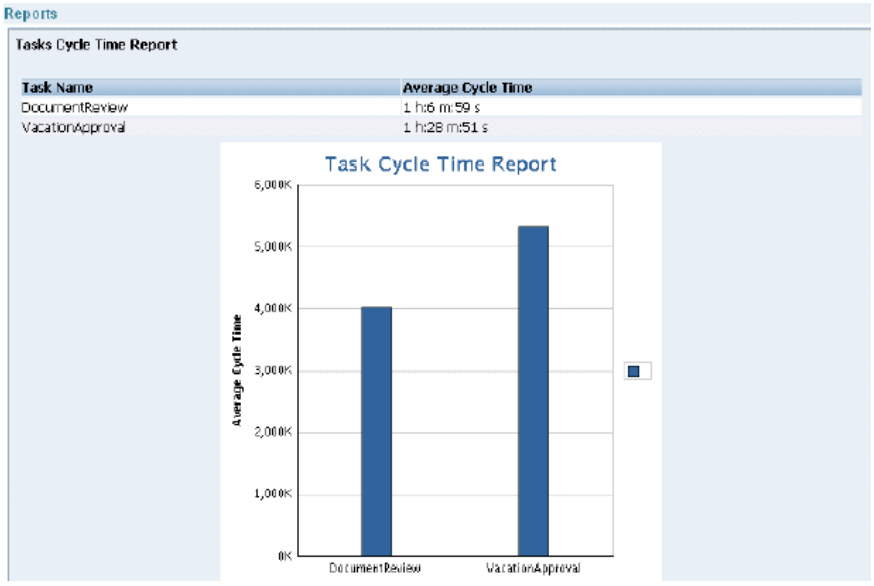


The report shows that the California group, the Supervisor group, and the LoanAgentGroup each have 16 tasks of normal priority. The users rsteven and jcooper have 5 and 22 tasks, respectively, all normal priority. Priorities (highest, high, normal, low, lowest) are distinguished by different colors in the bar chart.

29.11.2.3 Tasks Cycle Time Report

Figure 29–58 shows an example of a Tasks Cycle Time Report.

Figure 29–58 Tasks Cycle Time Report

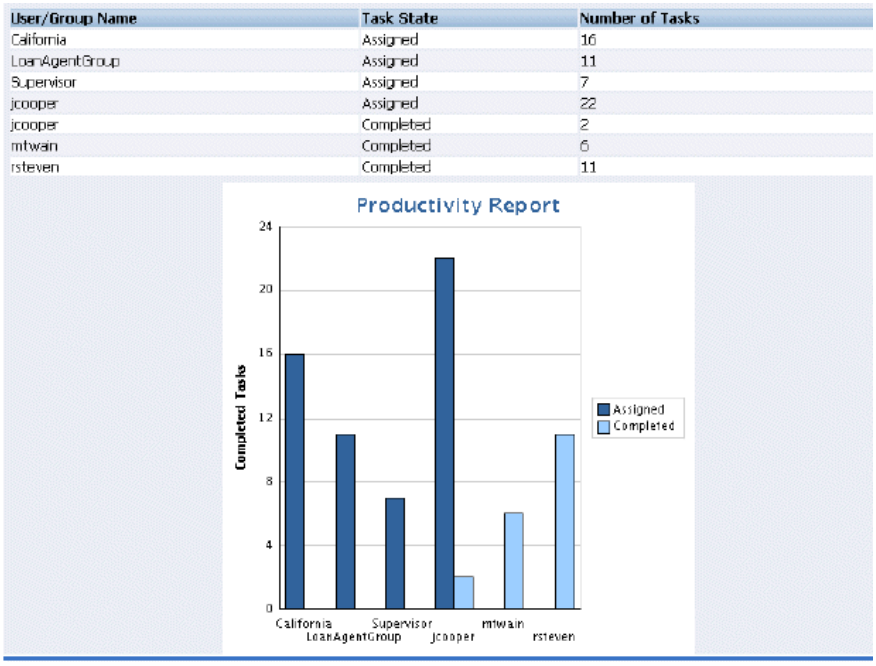


The report shows that it takes 1 hour and 6 minutes on average to complete DocumentReview tasks, and 1 hour and 28 minutes on average to complete VacationApproval tasks. The bar chart shows the average cycle time in milliseconds.

29.11.2.4 Tasks Productivity Report

Figure 29–59 shows an example of a Tasks Productivity Report.

Figure 29–59 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 default server locale is available, then the language is derived from the custom resource bundle class in 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 29.8.2, "How To Set the Worklist Display (Application Preferences)" 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 29.8.2, "How To Set the Worklist Display (Application Preferences)") or by providing an updated version of the default bundle class. See the Workflow Customizations sample for details.

For task attribute names, 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.

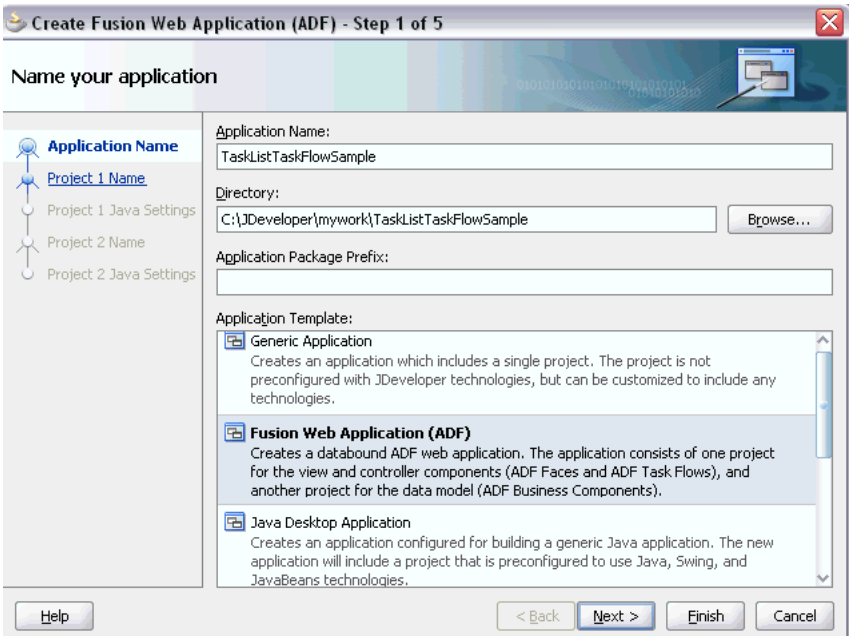
29.13.1 How to Create an Application With an Embedded Reusable Worklist Region

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 29–60 provides details.

Figure 29–60 Creation of Application with an Embedded Reusable Worklist Region

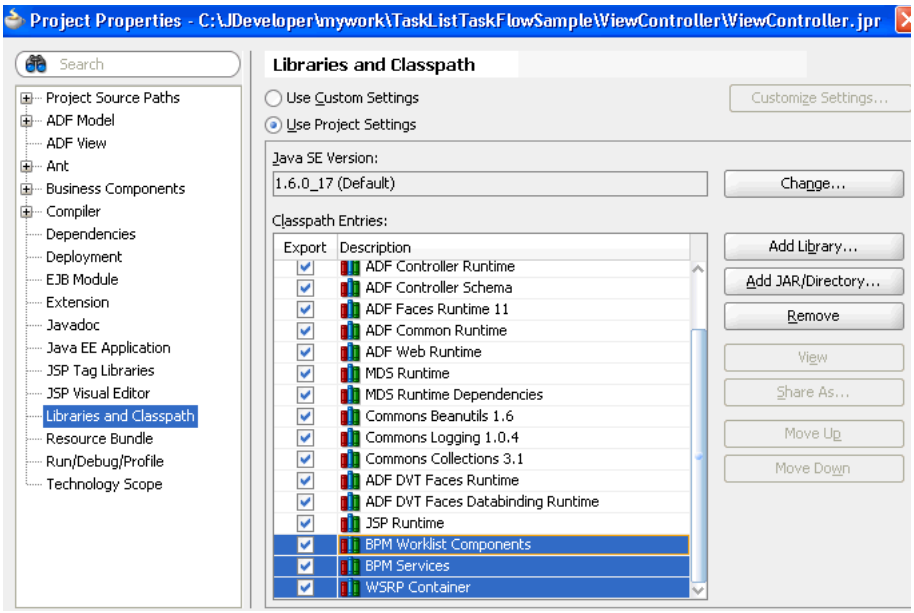


2. Open the View Controller Project Properties, **Libraries and Classpath** section, and click **Add Library** to add the following libraries in the class path:

   - **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 29–61 provides details.

Figure 29–61 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.

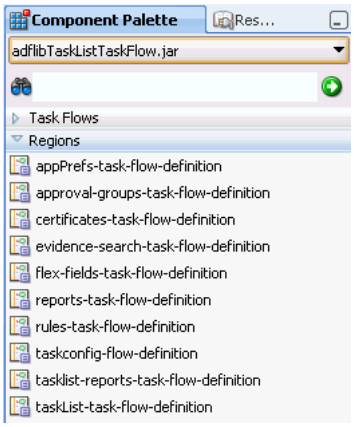
      Note that if you run the Task List task flow in federated mode, you do not need to do this step. See the section "federatedMode" on page 29-72 for information about how to use the task flow in federated mode.

4. Select the View Controller project and choose **File > New > Current Project Technologies > Web Tier > JSF Page** to create a jspx file (for example, `testSample.jspx`).

   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 29–62 provides details.

Figure 29–62 Component Palette



6. Drag and drop one of the task flow Regions to the jspx 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 *pagename*`Pagedef.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 29-72 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.

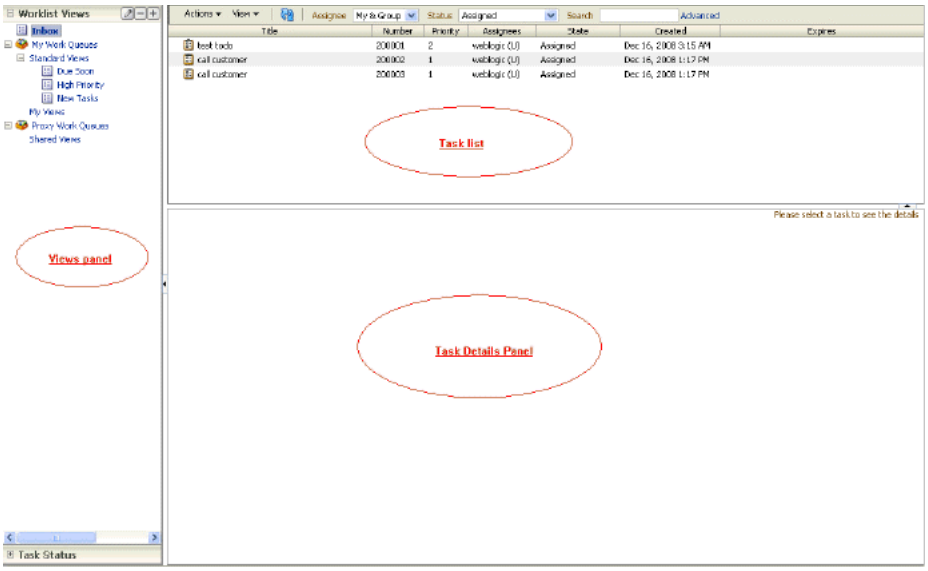
To restart the servers:

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 29–63 provides details.

Figure 29–63 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 29-72 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 to 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, your 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.WorkflowServicesClientConfigurationTyp
e;

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();
```

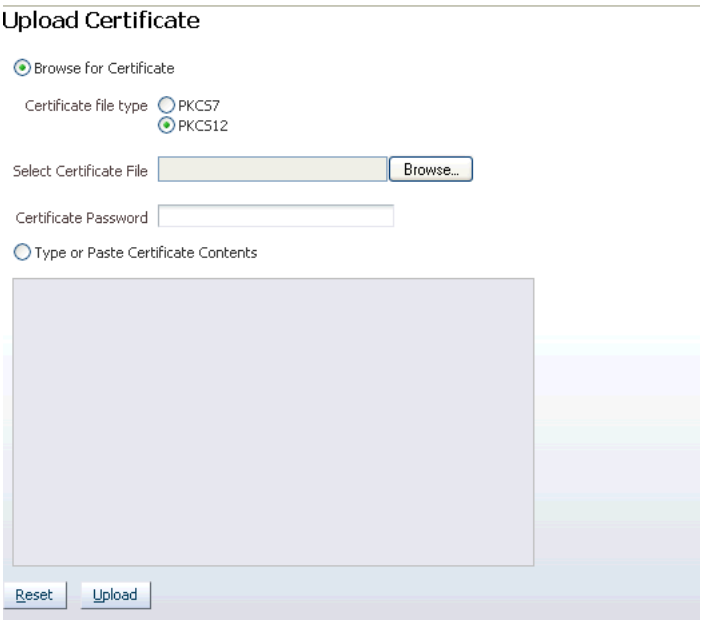
29.13.5 What You May Need to Know About Certificates Task Flow

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 29–64 provides details.

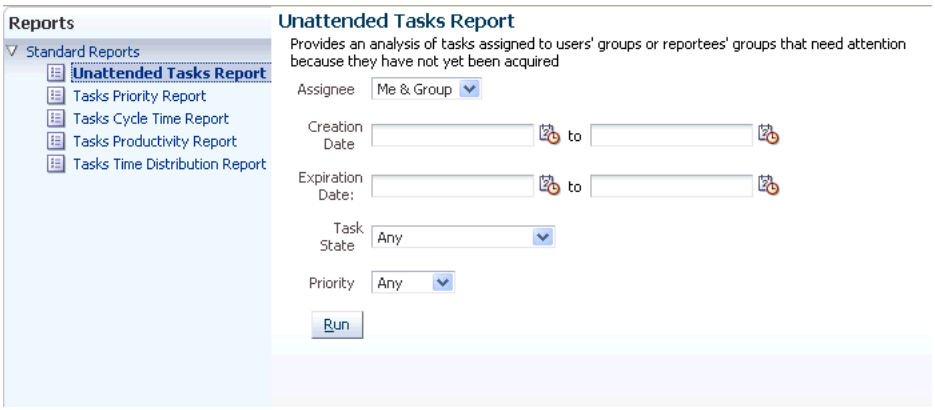
Figure 29–64 Digital Certificate



29.13.6 What You May Need to Know About the Reports Task Flow

Figure 29–65 shows the unattended tasks report.

Figure 29–65 Unattended Tasks Report



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 by 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.

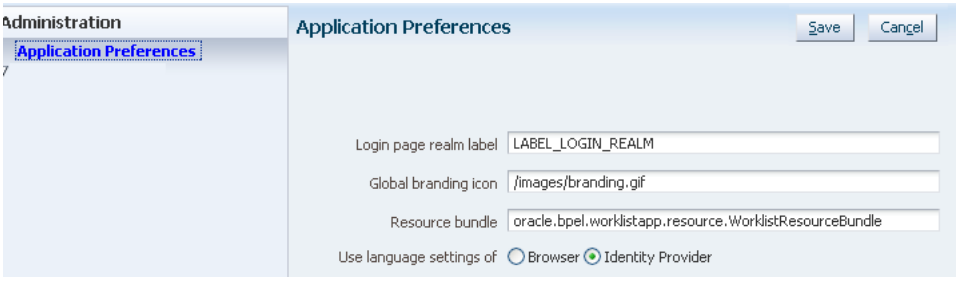
- **Task Type** - Use the Search (flashlight) icon to select from a list of task titles. All versions of a task are listed on the Select Workflow Task Type page (optional).

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 29–66 provides details.

Figure 29–66 Application Preferences



29.13.8 What You May Need to Know About Mapped Attributes Task Flow

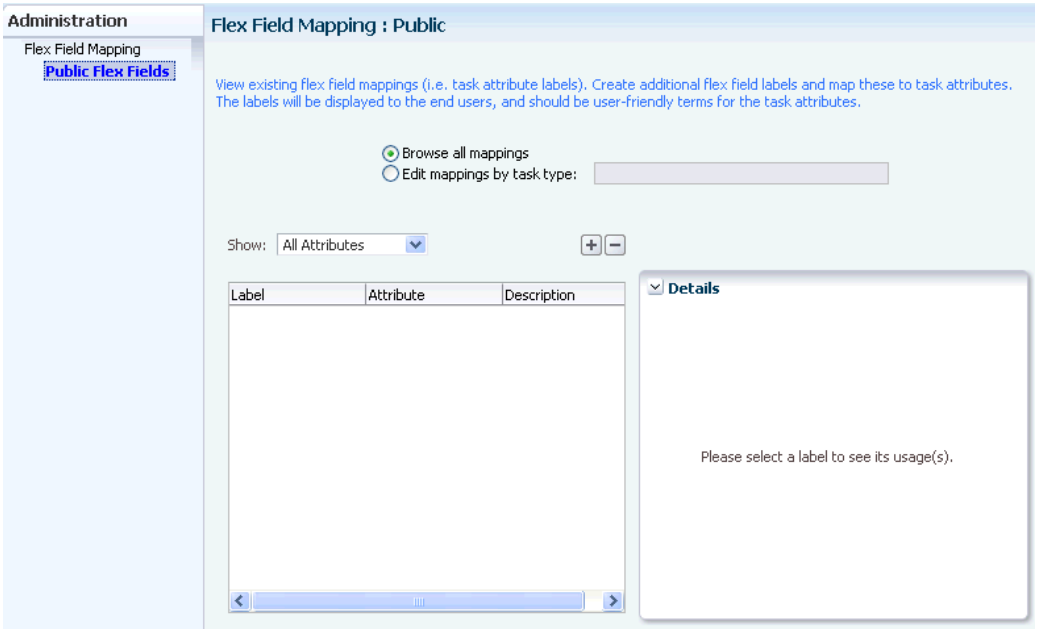
Human workflow mapped attributes store and query use case-specific custom attributes. These custom attributes typically come from the task payload values. Storing custom attributes in mapped attributes provides the following benefits:

- 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 29–67 provides details.

Figure 29–67 Mapped Attribute Mapping



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 29–68 and Figure 29–69 provide details.

Figure 29–68 Vacation Period

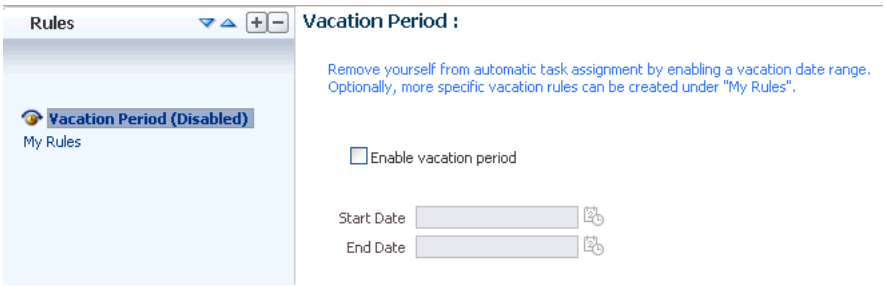
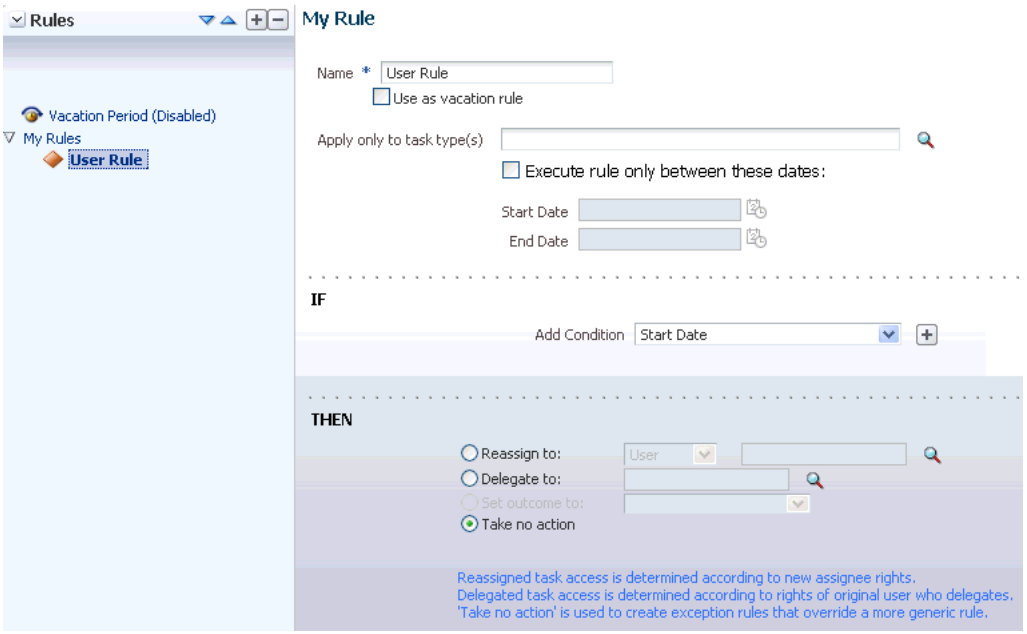


Figure 29–69 My Rule



30

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:

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 WorflowServiceClient
 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".toCharArry(),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
`getTaskService`	`oracle.bpel.services.workflow.task.ITaskService`
`getTaskQueryService`	`oracle.bpel.services.workflow.query.ITaskQueryService`
`getTaskReportService`	`oracle.bpel.services.workflow.report.ITaskReportService`

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 cellspacing="2" cellpadding="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 cellspacing="2" cellpadding="2" border="3" width="100%">
   <TR class="problemInfo">
       <TH>TaskNumber</TH>
       <TH>Title</TH>
       <TH>Priority</TH>
       <TH>CreatedDate</TH>
       <TH>Assignee(s)</TH>
       <TH>UpdatedDate</TH>
       <TH>UpdatedBy</TH>
       <TH>State</TH>
       <TH>Status</TH>
   </TR>
   <%
     SimpleDateFormat dflong = new SimpleDateFormat( "MM/dd/yy hh:mm a" );
     for(int i = 0 ; i < tasks.size() ; i ++)
     {
        Task task = (Task)tasks.get(i);
        int taskNumber = task.getSystemAttributes().getTaskNumber();
        String title = task.getTitle();
        int priority = task.getPriority();
        String assignee = getAssigneeString(task);
        Calendar createdDate = task.getSystemAttributes().getCreatedDate();
        Calendar updateDate =  task.getSystemAttributes().getUpdatedDate();
        String updatedBy = task.getSystemAttributes().getUpdatedBy().getId();
        String state = task.getSystemAttributes().getState();
        String outcome = task.getSystemAttributes().getOutcome();
        if(outcome == null) outcome = "";
        String titleLink = "http://" + request.getServerName() +
                           ":" + request.getServerPort() +
                            "/integration/worklistapp/TaskDetails?taskId=" +
                            task.getSystemAttributes().getTaskId();
```

```
     %>
       <tr class="userInfo">
          <td><%=taskNumber%></td>
          <td><a href="<%=titleLink%>" target="_blank"><%=title%></a></td>
          <td><%=priority%></td>
          <td><%=dflong.format(createdDate.getTime())%></td>
          <td><%=assignee%></td>
          <td><%=dflong.format(updateDate.getTime())%></td>
          <td><%=updatedBy%></td>
          <td><%=state%></td>
          <td><%=outcome%></td>
       <tr>
    <%
      }
    %>
 </table>
 </div>
 <%!
     private BPMAuthorizationService getAuthorizationService(String identityContext)
     {
      BPMAuthorizationService authorizationService =
ServiceFactory.getAuthorizationServiceInstance();
      if (identityContext != null)
        authorizationService = ServiceFactory.getAuthorizationServiceInstance(identityContext);

      return authorizationService;
     }
     private String getAssigneeString(Task task) throws Exception
     {
        List assignees = task.getSystemAttributes().getAssigneeUsers();
        StringBuffer buffer = null;
        for(int i = 0 ; i < assignees.size() ; i++)
        {
          IdentityType type = (IdentityType)assignees.get(i);
          String name = type.getId();
          if(buffer == null)
          {
             buffer = new StringBuffer();
          }
          else
          {
            buffer.append(",");
          }
          buffer.append(name).append("(U)");
        }
        assignees = task.getSystemAttributes().getAssigneeGroups();
        for(int i = 0 ; i < assignees.size() ; i++)
        {
          IdentityType type = (IdentityType)assignees.get(i);
          String name = type.getId();
          if(buffer == null)
          {
             buffer = new StringBuffer();
          }
          else
          {
            buffer.append(",");
          }
          buffer.append(name).append("(G)");
        }
```

```
         if(buffer == null)
         {
            return "";
         }
         else
         {
           return buffer.toString();
         }
      }
  %>
 </body>
</html>
```

31

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:

Note: In previous releases, Oracle BPM Worklist included a feature known as flex fields. Starting with Release 11*g* 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	`http://host:port/integration/services/TaskService/TaskServicePort?WSDL`
Task Query Service	`http://host:port/integration/services/TaskQueryService/TaskQueryService?WSDL`
Identity Service	`http://host:port/integration/services/IdentityService/configuration?WSDL`
	`http://host:port/integration/services/IdentityService/identity?WSDL`
Task Metadata Service	`http://host:port/integration/services/TaskMetadataService/TaskMetadataServicePort?WSDL`

Table 31–2 (Cont.) SOAP WSDL Location for the Task Services

Service name	SOAP WSDL location
User Metadata Service	`http://host:port/integration/services/UserMetadataService/UserMetadataService?WSDL`
Task Report Service	`http://host:port/integration/services/TaskReportService/TaskReportServicePort?WSDL`
Runtime Config Service	`http://host:port/integration/services/RuntimeConfigService/RuntimeConfigService?WSDL`
Evidence Store Service	`http://host:port/integration/services/EvidenceService/EvidenceService?WSDL`

Table 31–3 lists the JDNI 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	`ejb/bpel/services/workflow/TaskServiceBean`
Task Service Enterprise JavaBeans participating in client transaction	`ejb/bpel/services/workflow/TaskServiceGlobalTransactionBean`
Task Metadata Service	`ejb/bpel/services/workflow/TaskMetadataServiceBean`
Task Query Service	`ejb/bpel/services/workflow/TaskQueryService`
User Metadata Service	`ejb/bpel/services/workflow/UserMetadataService`
Runtime Config Service	`ejb/bpel/services/workflow/RuntimeConfigService`
Task Report Service	`ejb/bpel/services/workflow/TaskReportServiceBean`
Task Evidence Service	`ejb/bpel/services/workflow/TaskEvidenceServiceBean`

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
`acquireTask`	Acquire a task.
`acquireTasks`	Acquire a set of tasks.
`addAttachment`	Add an attachment to a task.
`addComment`	Add a comment to a task.
`createToDoTask`	Create a to-do task.
`delegateTask`	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.
`delegateTasks`	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.
`deleteTask`	Perform a logical deletion of a task. The task still exists in the database.
`deleteTasks`	Perform a logical deletion of a list of tasks. The tasks still exist in the database.
`errorTask`	Cause the task to error. This operation is typically used by the error assignee.
`escalateTask`	Escalate a task. The default escalation is to the manager of the current user. This can be overridden using escalation functions.
`escalateTasks`	Escalate tasks in bulk. The default escalation is to the manager of the current user. This can be overridden using escalation functions.
`getApprovers`	Get the previous approvers of a task.

Table 31–4 (Cont.) Task Service Methods

Method	Description
`getFutureParticipants`	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).
`getUsersToRequestInfoForTask`	Get the users from whom a request for information can be requested.
`initiateTask`	Initiate a task.
`mergeAndUpdateTask`	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
`overrideRoutingSlip`	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.
`purgeTask`	Remove a task from the persistent store.
`purgeTasks`	Remove a list of tasks from the persistent store.
`pushBackTask`	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.
`reassignTask`	Reassign a task.
`reassignTasks`	Reassign tasks in bulk.
`reinitiateTask`	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.
`releaseTask`	Release a previously acquired task.
`releaseTasks`	Release a set of previously acquired tasks.
`removeAttachment`	Remove a task attachment.
`renewTask`	Renew a task to extend the time it takes to expire.
`requestInfoForTask`	Request information for a task.

Table 31–4 (Cont.) Task Service Methods

Method	Description
`requestInfoForTaskWithReapproval`	Request information for a task with reapproval. For example, assume `jcooper` created a task and `jstein` and `wfaulk` approved the task in the same order. When the next approver, `cdickens`, requests information with reapproval from `jcooper`, and `jcooper` submits the information, `jstein` and `wfaulk` approve the task before it comes to `cdickens`. If `cdickens` requests information with reapproval from `jstein`, and `jstein` submits the information, `wfaulk` approves the task before it comes to `cdickens`.
`resumeTask`	Resume a task. Operations can only be performed by the task owners (or users with the `BPMWorkflowSuspend` privilege) to remove the hold on a workflow. After a human workflow is resumed, actions can be performed on the task.
`resumeTasks`	Resume a set of tasks.
`routeTask`	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.
`skipCurrentAssignment`	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.
`submitInfoForTask`	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.
`suspendTask`	Allow task owners (or users with the `BPMWorkflowSuspend` 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).
`suspendTasks`	Suspend a set of tasks.
`updateOutcomeOfTasks`	Update the outcome of a set of tasks.
`updateTask`	Update the task.
`updateTaskOutcome`	Update the task outcome.
`updateTaskOutcomeAndRoute`	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.
`withdrawTask`	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 `Withdrawn`.
`withdrawTasks`	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 SOA Suite*
- 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
`authenticate`	Authenticates a user with the identity authentication service and passes back a valid `IWorkflowContext` object.
`authenticateOnBehalfOf`	Optionally makes authentication on behalf of another user.
`countTasks`	Counts the number of tasks that match the specified query criteria.
`countViewTasks`	Counts the number of tasks that match the query criteria of the specified view.
`createContext`	Creates a valid `IWorkflowContext` object from a preauthenticated HTTP request.
`doesTaskExist`	Checks to see if any existing tasks match the specified query criteria.
`doesViewTaskExist`	Checks to see if any tasks exist match the query criteria of the specified view.
`getWorkflowContext`	Gets a human workflow context with the specified context token.
`destroyWorkflowContext`	Cleans up a human workflow context that is no longer needed. This method is typically used when a user logs out.
`getTaskDetailsById`	Gets the details of a specific task from the task's `taskId` property.
`getTaskDetailsByNumber`	Gets the details of a specific task from the task's `task number` property.
`getTaskHistory`	Gets a list of the task versions for the specified task ID.
`getTaskSequence`	Gets the task sequence tree of a task whose ID is a task ID, for those type of sequences.
`getTaskVersionDetails`	Gets the specific task version details for the specified task ID and version number.
`getWorkflowContextForAuthenticatedUser`	Gets the `IWorkflowContext` object for a user authenticated by a JAAS application. Use this either with Enterprise JavaBeans or SAML token identity propagation.
`queryAggregatedTasks`	Executes the specified query, and aggregates a count of the tasks returned by the query, grouped by the specified column.
`queryTaskErrors`	Returns a list of task error objects matching the specified predicate.

Table 31–5 (Cont.) Task Query Service Methods

Method	Description
`queryTasks`	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: ■ *`assignmentFilter`*: 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. `REPORTEES`: Returns tasks that are assigned to reportees of the context user. `REVIEWER`: Returns tasks for which the context user is a reviewer. ■ *`keywords`*: 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). ■ *`predicate`*: An optional `oracle.bpel.services.workflow.repos.Predicate` object that allows clients to specify complex, SQL-like query predicates.
`queryViewAggregatedTasks`	Executes the query as defined in the specified view, and aggregates the selected tasks according to the chart property defined in the view.
`queryViewTasks`	Returns a list of tasks according to the criteria in the specified view. The entire list or paged list of tasks can be returned. Clients can specify additional filter and ordering criteria to those in the view.

For more information, see the following:

- Section 31.1.11, "Task Instance Attributes"
- *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle SOA Suite* 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.

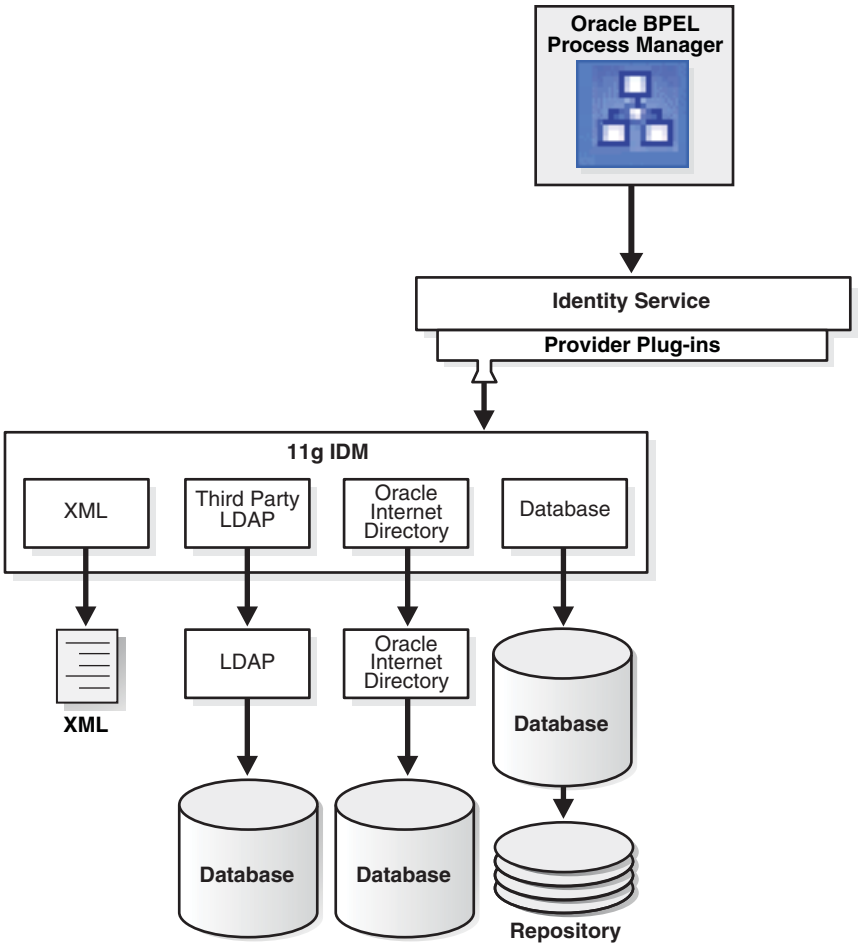
For more information, see the following:

- *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 11*g*, 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 11*g*, 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
`getTaskMetadataByNamespace`	Gets the `TaskMetadata` object that describes the human task service component with the specified task definition namespace and composite version.
`getOutcomes`	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
`getResourceBundleInfo`	Gets the resource bundle information of the task. The resource bundle information contains the location and the name of the bundle.
`getRestrictedActions`	Gets the actions that are restricted for a particular task.
`getTaskAttributesForTaskDefinitions`	Gets a list of `TaskAttribute` objects that describe standard task attributes and mapped attribute columns that are common for the specified task definitions.
`getTaskAttributesForTaskNamespaces`	Gets a list of `TaskAttribute` objects that describe standard task attributes and mapped attribute columns that are common for task definitions identified by the specified namespaces.
`getTaskAttributes`	Gets the task message attributes.
`getTaskAttributesForTaskDefinition`	Gets the message attributes for a particular task definition.
`getTaskDefinition`	Gets the task definition associated with the task.
`getTaskDefinitionById`	Gets the task definition by the task definition ID.
`getTaskDefinitionOutcome`	Gets the outcomes given the task definition ID.
`getTaskDisplay`	Gets the task display for a task.
`getTaskVisibilityRules`	Gets the task visibility rules.
`getTaskDisplayRegion`	Gets the task display region for a task.
`getVersionTrackedAttrs`	Gets the task attributes that when changed cause a task version creation.
`listTaskMetadata`	Lists the task definitions in the system.

For more information, see *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle SOA Suite*.

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
`createRule`	Creates a new rule.
`decreaseRulePriority`	Decreases the priority of a rule by one. This method does nothing if this rule has the lowest priority.
`deleteRule`	Deletes a rule.
`getVacationInfo`	Retrieves the date range (if any) during which a user is unavailable for the assignment of tasks.
`getRuleDetail`	Gets the details for a particular human workflow rule.
`getRuleList`	Retrieves a list of rules for a particular user or group.
`updateRule`	Updates an existing rule.
`increaseRulePriority`	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.
`getUserTaskViewList`	Gets a list of the user task views that the user owns.
`getGrantedTaskViewList`	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.
`getStandardTaskViewList`	Gets a list of standard task views that ship with the human workflow service, and are available to all users.
`getUserTaskViewDetails`	Gets the details for a single view.
`createUserTaskView`	Creates a new user task view.
`updateUserTaskView`	Updates an existing user task view.
`deleteUserTaskView`	Deletes a user task view.
`updateGrantedTaskView`	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.
`getUserPreferences`	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 `workflow.admin` privilege).
`setUserPreferences`	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.
`getPublicPreferences`	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 `workflow.admin` privilege, can update them. Public preferences are useful for storing application-wide preferences (preferences can be stored under a dummy user name, such as `MyAppPrefs`).
`setPublicPreferences`	Sets the public preferences for the user.

Table 31–7 (Cont.) User Metadata Service Methods

Method	Description
`setVacationInfo`	Sets a date range over which the user is unavailable for the assignment of tasks. (Dynamic assignment functions do not assign tasks to a user that is on vacation.)
`getStandardTaskView Details`	Gets the full details for a particular standard view, identified by its `viewId`.

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 SOA Suite*

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
`CreateAttributeLabel`	Creates a new attribute label for a particular task mapped attribute.
`createPayloadMapping`	Creates a new mapping between an attribute label and a task payload field.
`DeleteAttributeLabel`	Deletes an existing attribute label.
`deletePayloadMapping`	Deletes an existing payload mapping.
`getAttributeLabelUsages`	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.
`getGroupDynamicAssignmentFunctions`	Returns a list of the dynamic assignment functions that can select a group that are implemented on this server.
`getTaskDisplayInfo`	Retrieves information relating to the URIs used for displaying task instances of a specific task definition.
`getTaskStatus`	Gets the status of a task instance corresponding to a particular task definition and composite instance.
`getUserDynamicAssignmentFunctions`	Returns a list of the dynamic assignment functions that can select a user that are implemented on this server.
`GetWorkflowPayloadMappings`	Gets a list of all the mapped attribute mappings for a particular human workflow definition.
`setTaskDisplayInfo`	Sets information relating to the URIs to be used for displaying task instances of a specific task definition.
`updateAttributeLabel`	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 SOA Suite*

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

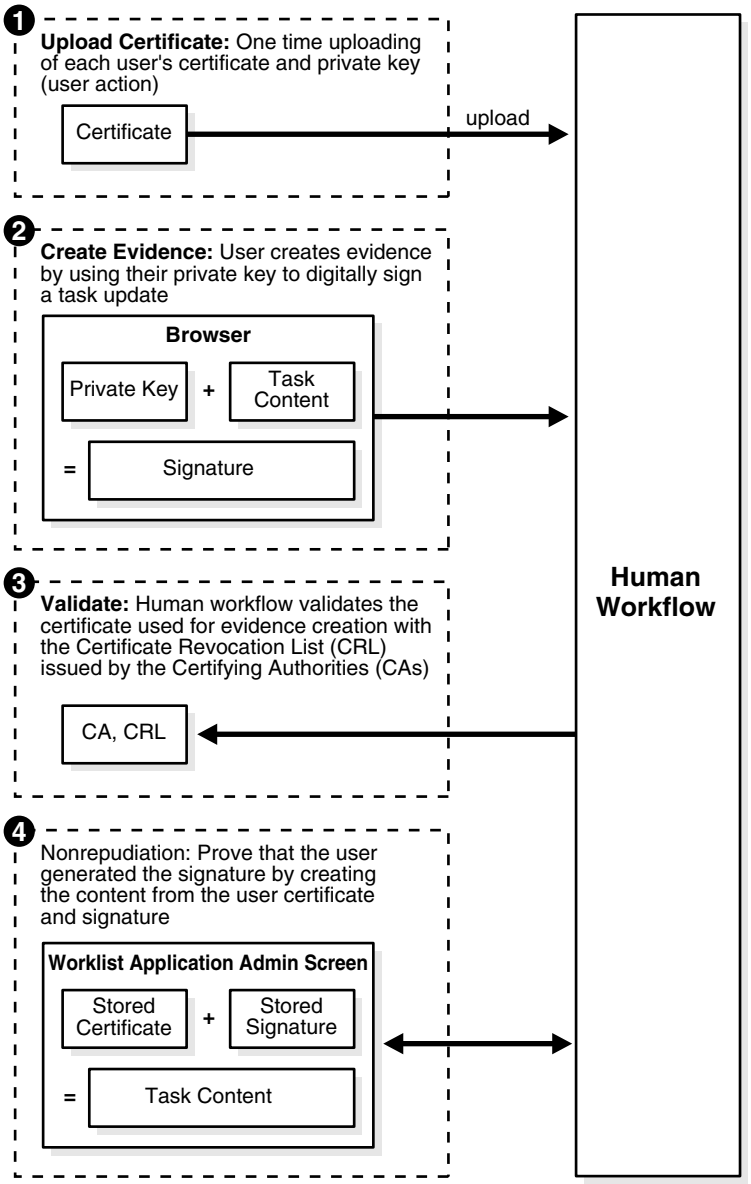
A cryptographically-based digital signature is created when a public key algorithm signs a sender's message with a sender's private key.

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
`createEvidence`	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 `workflow.admin.evidenceStore` 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 `true` is returned if the signature is verified. Otherwise, `false` 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 `PASSWORD` or `CERTIFICATE`.
`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: ■ `AVAILABLE`: The evidence is available for CRL validation. ■ `FAILURE`: CRL validation failed. ■ `SUCCESS`: CRL validation succeeded. ■ `UNAVAILABLE`: The CRL data could not be fetched. ■ `WAIT`: CRL validation is in 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
`getTaskVersion`	Gets the version of the task with which this evidence is associated.
`getVerifiedDate`	Gets the date on which the CRL validation of the certificate used was performed.
`getWorkflowType`	Gets the workflow type of the task with which this evidence is associated. This is typically `BPELWF`.

Table 31–13 Certificate Interface

Method	Description
`getCA`	Gets the certificate issuer's distinguished name (DN).
`getCertificate`	Gets the certificate object that is abstracted by the interface.
`getID`	Gets the certificate's serial number.
`getIdentityContext`	Gets the identity context with which the uploader of this certificate is associated.
`getUserName`	Gets the user name with whom this certificate is associated.
`isValid`	Returns `true` if the certificate is still valid.

Table 31–14 Policy Type and Workflow Type Interface

Method	Description
`fromValue`	Constructs an object from the string representation.
`value`	Returns the string representation of this object.

For more information, see the following:

- Section 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
`task/applicationContext`	The application with which any application roles associated with this task (assignees, owners, and so on) belong.
`task/category`	An optional category of the task.
`task/creator`	The name of the creator of this task.
`task/dueDate`	The due date for the task. This is used on to-do tasks.

Table 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/updatedDate`	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/updatedBy`	The user who added the comment.
`task/userComment/updatedDate`	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 /subTaskGroupInstance Id`	A unique ID that is set on a subtask. This same ID is set on the parent task's `taskGroupInstanceId`. 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 /taskGroupInstanceId`	A unique ID that is set on the parent task. This same ID is set on the subtask's `subTaskGroupInstanceId`. 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 /updatedDate`	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/systemMessageAtt ributes/*`	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 11*g*, 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 11*g*, 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.

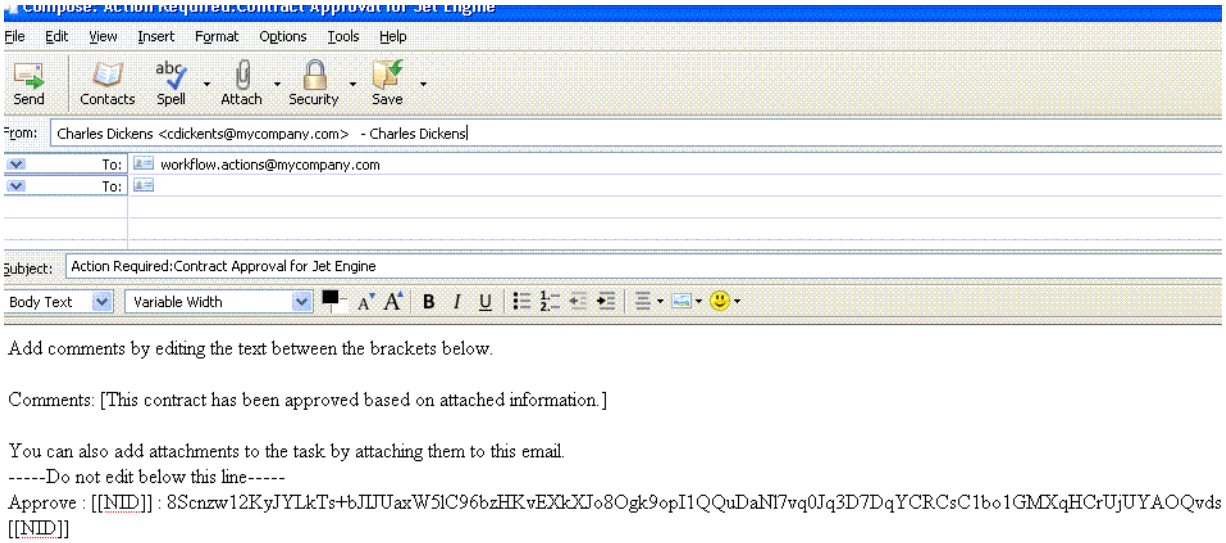
   If a notification is actionable, the email contains links for each of the custom outcomes.

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

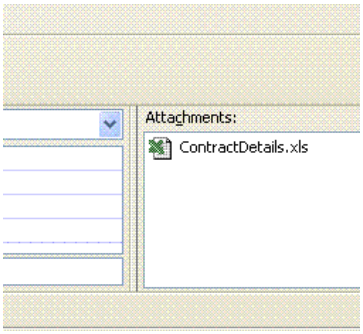


4. Add comments in the comments section of the approval mail. For example:

   ```
   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 sufixed to all auto reply mesages
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 SOA Suite*.

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, `jlondon`.
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`, `updatedDate`, 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.routingslip.model.*;
```

```
import oracle.bpel.services.workflow.metadata.routingslip.model.Participants;
import
oracle.bpel.services.workflow.metadata.routingslip.model.ParticipantsType.*;
import oracle.bpel.services.workflow.task.IAssignmentService;
import oracle.bpel.services.workflow.task.ITaskAssignee;
import oracle.bpel.services.workflow.task.model.Task;
public class TestAssignmentService implements
 oracle.bpel.services.workflow.task.IAssignmentService {
    static int numberOfApprovals = 0;
    static String[] users = new String[]{"jstein", "wfaulk", "cdickens"};
    public Participants onInitiation(Task task,
                                     Map propertyBag) {
        return createParticipant();
    }
    public Participants onReinitiation(Task task,
                                       Map propertyBag) {
        return null;
    }
    public Participants onOutcomeUpdated(Task task,
                                         Map propertyBag,
                                         String updatedBy,
                                         String outcome) {
        return createParticipant();
    }
    public Participants onAssignmentSkipped(Task task,
                                            Map propertyBag) {
        return null;
    }
    public List getAssigneesToRequestForInformation(Task task,
                                                    Map propertyBag) {
        List rfiUsers = new ArrayList();
        rfiUsers.add("jcooper");
        rfiUsers.add("jstein");
        rfiUsers.add("wfaulk");
        rfiUsers.add("cdickens");
        return rfiUsers;
    }
    public List getReapprovalAssignees(Task task,
                                       Map propertyBag,
                                       ITaskAssignee infoRequestedAssignee) {
        List reapprovalUsers = new ArrayList();
        reapprovalUsers.add("jstein");
        reapprovalUsers.add("wfaulk");
        reapprovalUsers.add("cdickens");
        return reapprovalUsers;
    }
    private Participants createParticipant() {
        if (numberOfApprovals > 2) {
            numberOfApprovals = 0;
            return null;
        }
        String user = users[numberOfApprovals++];

        ObjectFactory objFactory = new ObjectFactory();
        Participants participants = objFactory.createParticipants();
        Participant participant = objFactory.createParticipantsTypeParticipant();
        participant.setName("Loan Agent");
        ResourceType resource2 = objFactory.createResourceType(user);
        resource2.setIsGroup(false);
        resource2.setType("STATIC");
```

```
        participant.getResource().add(resource2);

        participants.getParticipantOrSequentialParticipantOrAdhoc().
          add(participant);
        return participants;
    }

}
```

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 `worklfowCustomClasspathURL` 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 `worklfowCustomClasspathURL` 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 wscc, 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

```
/**
 * WFClientSample
 */
package oracle.bpel.services.workflow.samples;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

import oracle.bpel.services.workflow.IWorkflowConstants;
import oracle.bpel.services.workflow.WorkflowException;
import oracle.bpel.services.workflow.client.IWorkflowServiceClient;
import oracle.bpel.services.workflow.client.WorkflowServiceClientFactory;
import oracle.bpel.services.workflow.client.IWorkflowServiceClientConstants
 .CONNECTION_PROPERTY;
import oracle.bpel.services.workflow.query.ITaskQueryService;
import oracle.bpel.services.workflow.query.ITaskQueryService.AssignmentFilter;
import oracle.bpel.services.workflow.query.ITaskQueryService.OptionalInfo;
import oracle.bpel.services.workflow.repos.Ordering;
import oracle.bpel.services.workflow.repos.Predicate;
import oracle.bpel.services.workflow.repos.TableConstants;
import oracle.bpel.services.workflow.verification.IWorkflowContext;

public class WFClientSample {

  public static List  runClient(String clientType) throws WorkflowException {
      try {

         IWorkflowServiceClient wfSvcClient = null;
         ITaskQueryService taskQuerySvc = null;
         IWorkflowContext wfCtx = null;

         // 1. this step is optional since configuration can be set in  wf_client_
             config.xml file
         Map<CONNECTION_PROPERTY, String> properties = new HashMap<CONNECTION_
PROPERTY, String>();
         if (WorkflowServiceClientFactory.REMOTE_CLIENT.equals(clientType)) {
           properties.put(CONNECTION_PROPERTY.EJB_INITIAL_CONTEXT_FACTORY,
 "weblogic.jndi.WLInitialContextFactory");
           properties.put(CONNECTION_PROPERTY.EJB_PROVIDER_URL,
 "t3://myhost.us.oracle.com:7001");
           properties.put(CONNECTION_PROPERTY.EJB_SECURITY_CREDENTIALS,
 "weblogic");
           properties.put(CONNECTION_PROPERTY.EJB_SECURITY_PRINCIPAL, "weblogic");
         } else if (WorkflowServiceClientFactory.SOAP_CLIENT.equals(clientType)) {
           properties.put(CONNECTION_PROPERTY.SOAP_END_POINT_ROOT,
 "http://myhost:7001");
           properties.put(CONNECTION_PROPERTY.SOAP_IDENTITY_
PROPAGATION,"non-saml"); // optional
         }
         // 2. gets IWorkflowServiceClient for specified client type
         wfSvcClient =
 WorkflowServiceClientFactory.getWorkflowServiceClient(clientType, properties,
 null);

         // 3. gets ITaskQueryService instance
         taskQuerySvc = wfSvcClient.getTaskQueryService();

         // 4. gets IWorkflowContext instance
         wfCtx = taskQuerySvc.authenticate("jcooper", "welcome1".toCharArray(),
 "jazn.com");

         // 5. creates displayColumns
         List<String> displayColumns = new ArrayList<String>(8);
         displayColumns.add("TASKID");
         displayColumns.add("TASKNUMBER");
         displayColumns.add("TITLE");
```

```
        displayColumns.add("CATEGORY");

        // 6. creates optionalInfo
        List<ITaskQueryService.OptionalInfo> optionalInfo = new
ArrayList<ITaskQueryService.OptionalInfo>();
        optionalInfo.add(ITaskQueryService.OptionalInfo.DISPLAY_INFO);

        // 7. creates assignmentFilter
        AssignmentFilter assignmentFilter = AssignmentFilter.MY_AND_GROUP;

        // 8. creates predicate
        List<String> stateList = new ArrayList<String>();
        stateList.add(IWorkflowConstants.TASK_STATE_ASSIGNED);
        stateList.add(IWorkflowConstants.TASK_STATE_INFO_REQUESTED);
        Predicate predicate = new Predicate(TableConstants.WFTASK_STATE_COLUMN,
Predicate.OP_IN, stateList);

        // 9. creates ordering
        Ordering ordering = new Ordering(TableConstants.WFTASK_DUEDATE_COLUMN,
true, false);
        ordering.addClause(TableConstants.WFTASK_CREATEDDATE_COLUMN, true,
false);

        // 10. calls service - query tasks
        List taskList = taskQuerySvc.queryTasks(wfCtx,
                                                (List<String>) displayColumns,
                                                (List<OptionalInfo>) optionalInfo,
                                                (AssignmentFilter)
                                                  assignmentFilter,
                                                (String) null, // keywords is
optional (see javadoc)
// optional
                                                 predicate,
                                                 ordering,
                                                 0,    // starting row
                                                 100); // ending row for paging, 0
                                                    if no paging

        // Enjoy result
        System.out.println("Successfuly 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 wscc,
                          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 wscc` 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) hrows 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, // dafault 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 11*g* 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 11*g* Release 1. To migrate from release 10.1.3.*x* to 11*g* 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<ONNECTION_
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(ONNECTION_PROPERTY.SECURITY_POLICY_URI, "oracle/wss10_saml_token_
client_policy"); //optional
properties.put(CONNECTION_PROPERTY.MANAGEMENT_POLICY_URI , "oracle/log_policy");
  //optional
IWorkflowServiceClient client =
                           WorkflowServiceClientFactory.getWorkflowServiceClient(
                           WorkflowServiceClientFactory.SOAP_CLIENT,
properties, null);
```

The client reference to the policy URI must match the server policy URI. Otherwise, SAML token propagation fails.

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<ONNECTION_
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(ONNECTION_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 11*g* 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 SOA Suite*.

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;
```

32

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:

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-c
lass>
</filter>
<filter>
<filter-name>ADFLibraryFilter</filter-name>
<filter-class>oracle.adf.library.webapp.LibraryFilter</filter-class>
```

```
</filter>
...
<filter-mapping>
<filter-name>adfdiExcelDownload</filter-name>
<url-pattern>*.xlsx</url-pattern>
</filter-mapping>
<filter-mapping>
<filter-name>adfdiExcelDownload</filter-name>
<url-pattern>*.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

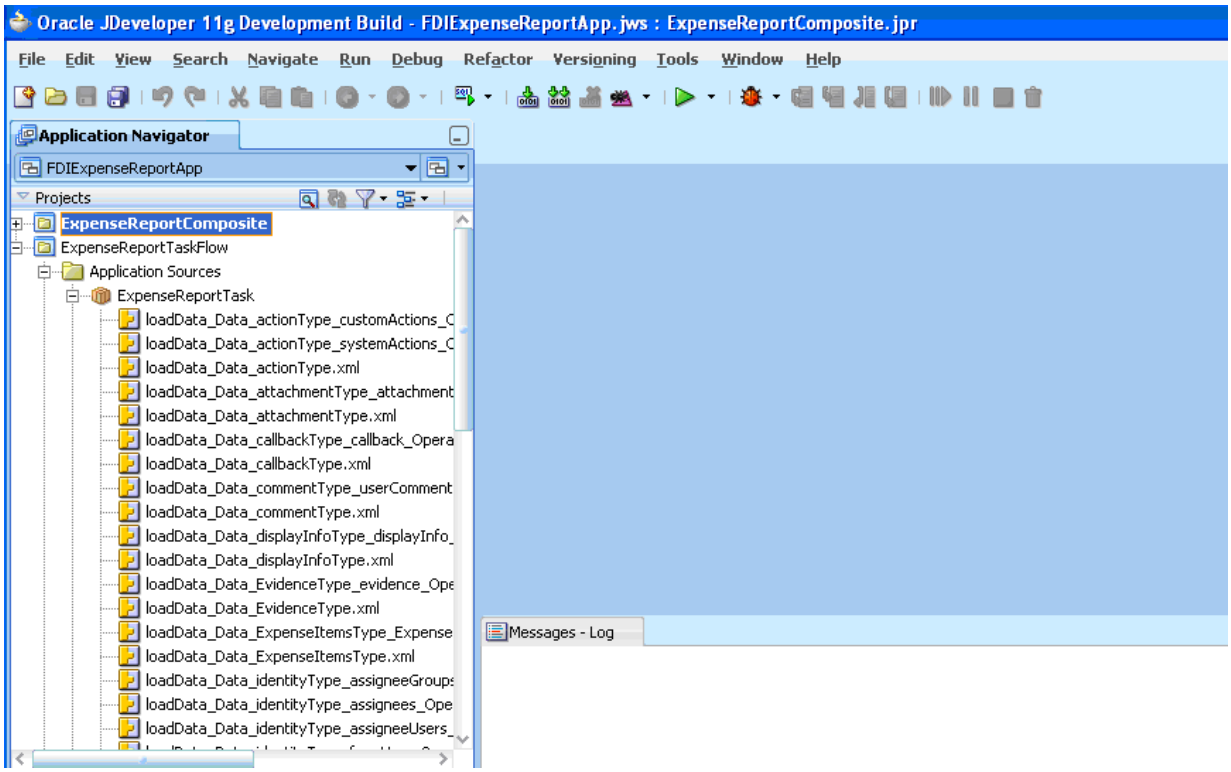
This section describes how to attach an Excel workbook to email notifications.

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.

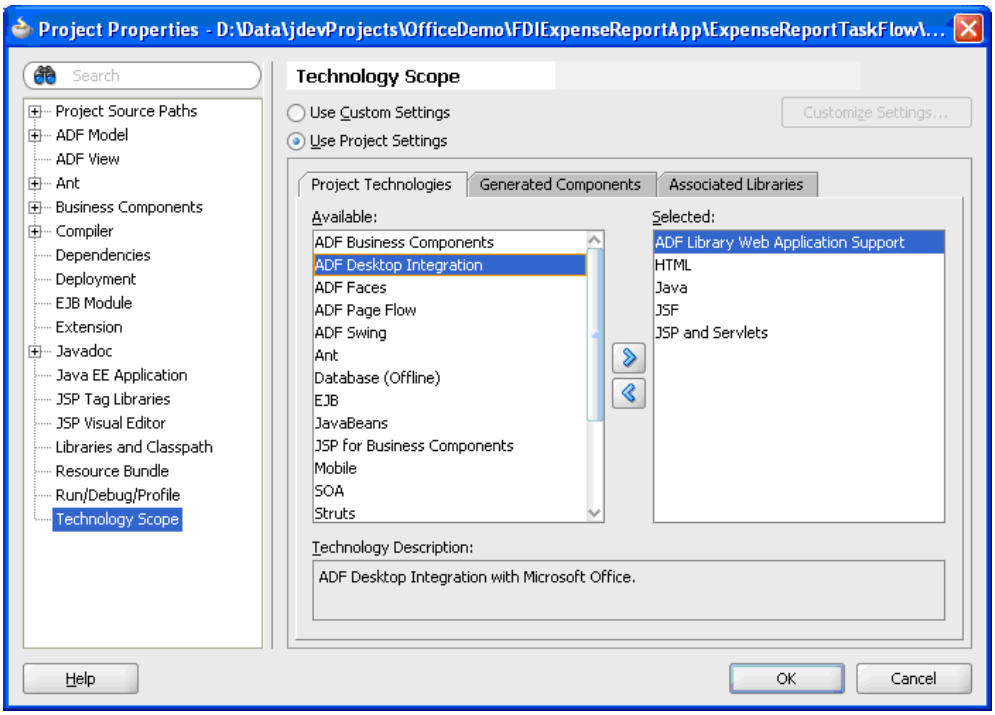
Figure 32–1 Oracle JDeveloper with a Sample Project Open



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.

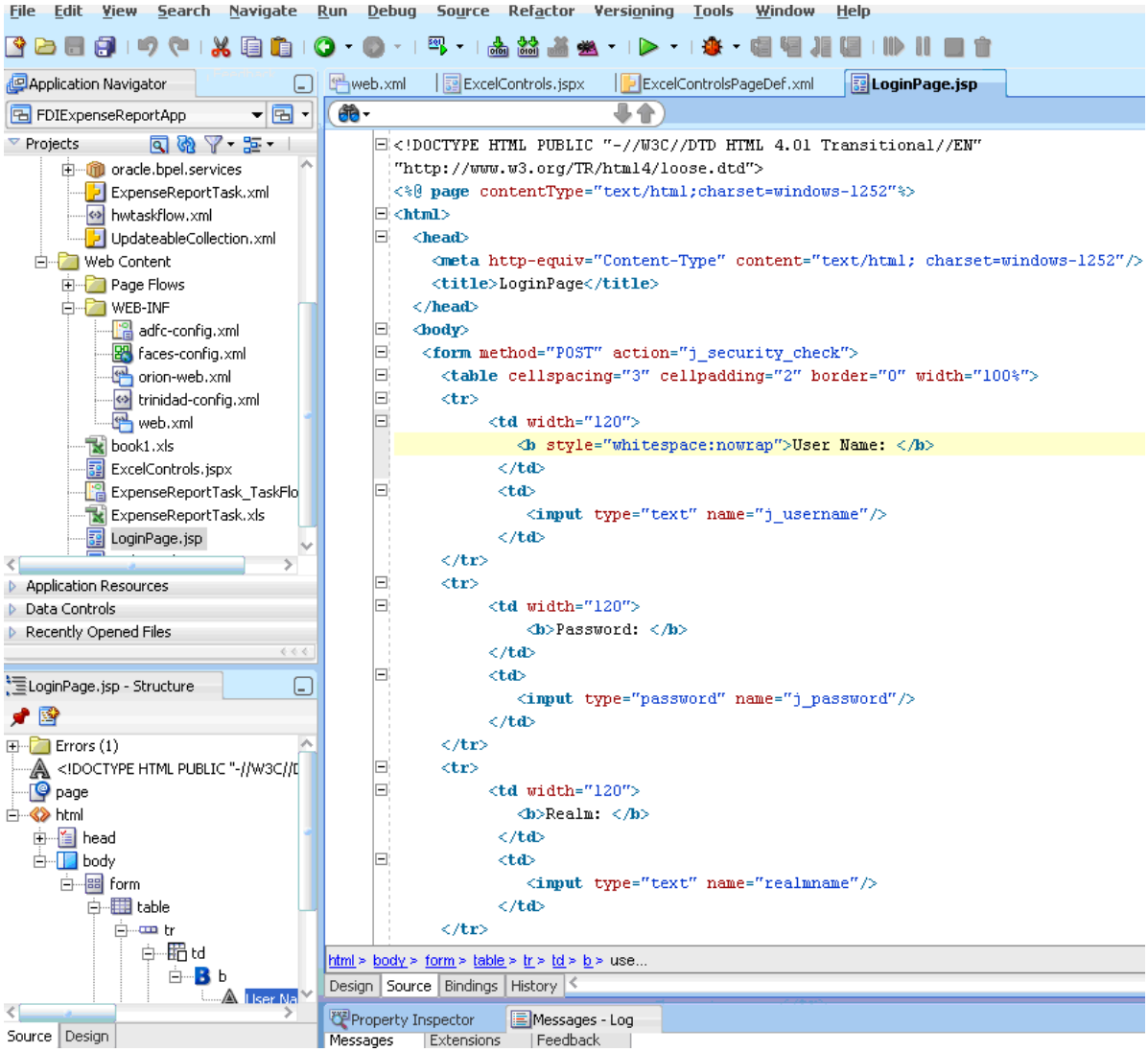
The web application is now configured to work with Oracle ADF-DI.

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



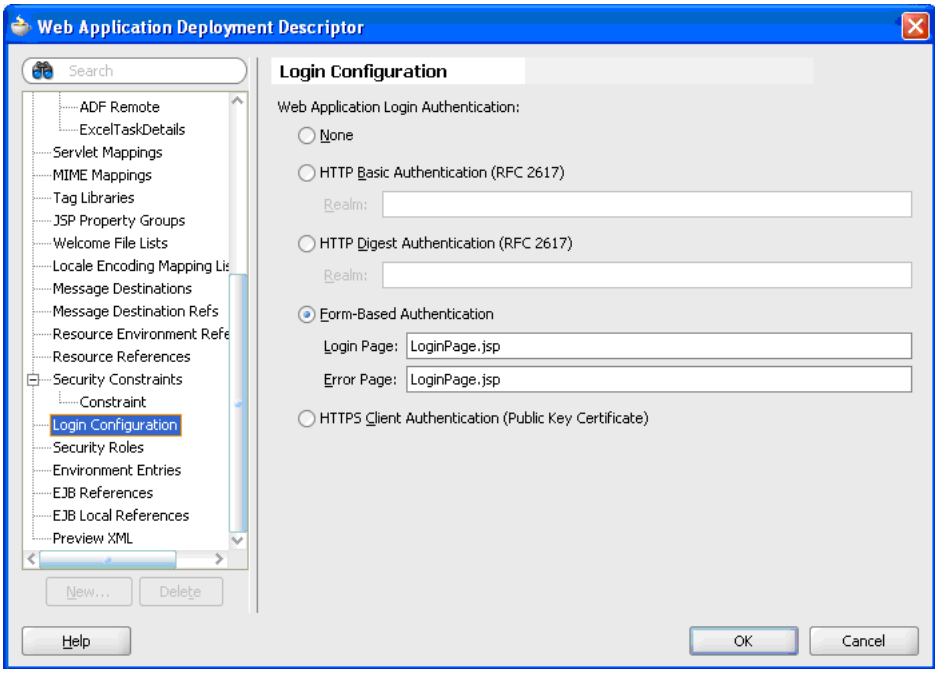
3. Once you have added ADF security, confirm that the following entries are added to the `web.xml` file. If some entries are missing, add them manually. Note that form authentication, using the login page created in Step 2 of Section 32.2.3.2, "Task 2: Set up Authentication," is used.

```
<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:

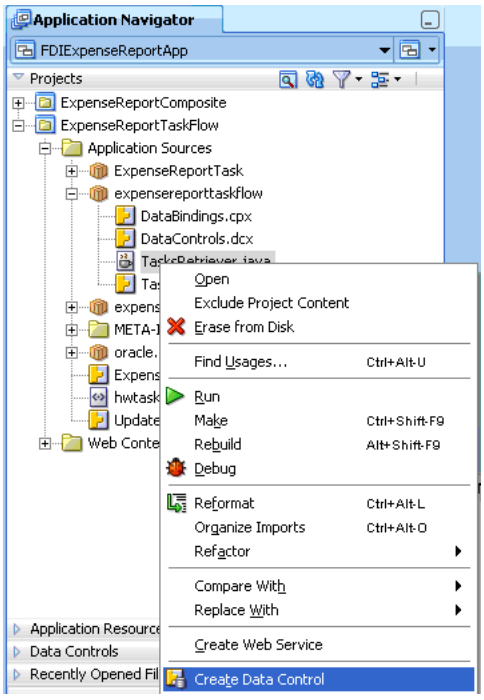
      Name: `TaskRetrievers`

      Package: (leave it as default)

      Extends: oracle.bpel.services.workflow.client.worklist.excel.TasksRetriever (Click **Browse** to select this class.)

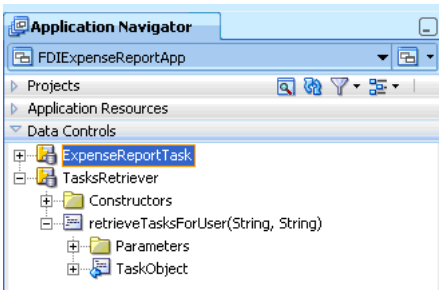
      This creates a new Java class `<default-package>.TasksRetriever`.
2. Create a data control for this newly created Java class. This data control provides access to an API that retrieves all assigned tasks for a user. Figure 32–5 shows the menu for creating the data control.

Figure 32–5 Oracle JDeveloper: Creating a Data Control



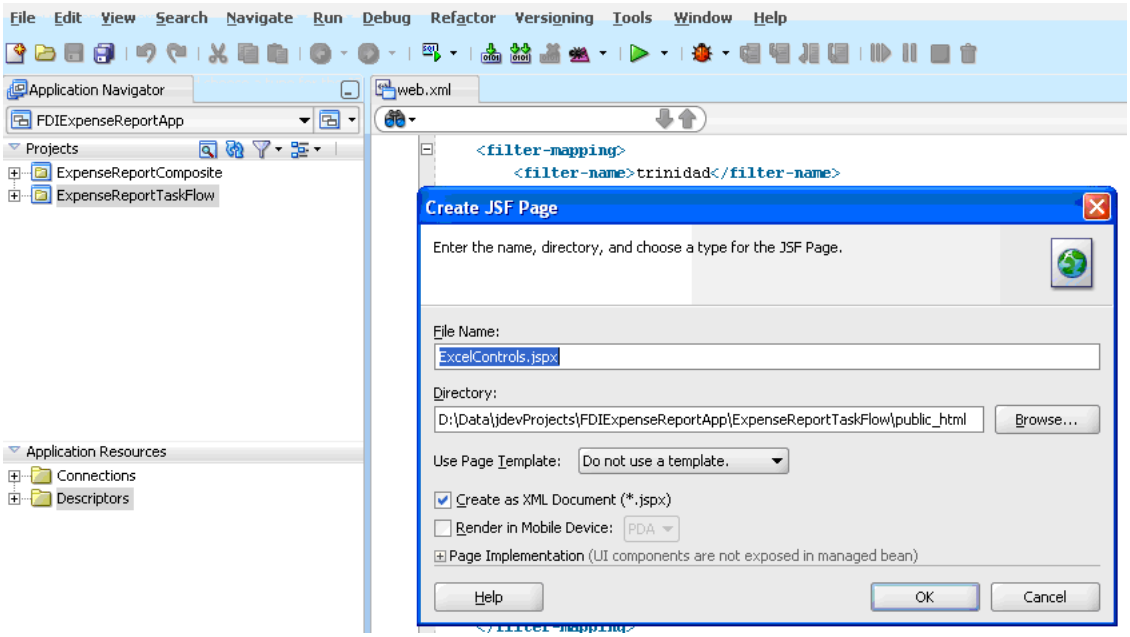
3. Verify that the newly created data control **TasksRetriever** is visible in the Data Control palette in the lower portion of the Application Navigator. Figure 32–6 shows the Application Navigator with the Data Control palette expanded.

Figure 32–6 Oracle JDeveloper: Application Navigator with Data Control Palette Expanded



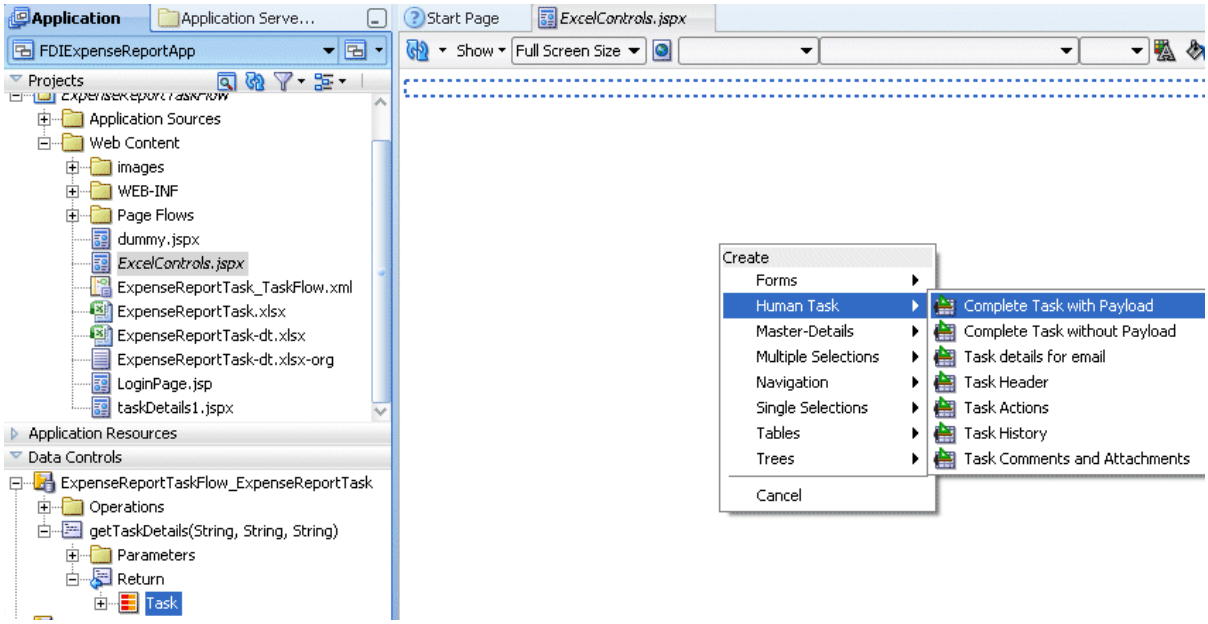
4. Create a new JSF JSP page--namely, `ExcelControls.jspx`. This generates a page definition that can be used by ADF-DI while authoring the Excel document. Figure 32–7 provides details.

Figure 32–7 Oracle JDeveloper: Creating a JSF JSP Page



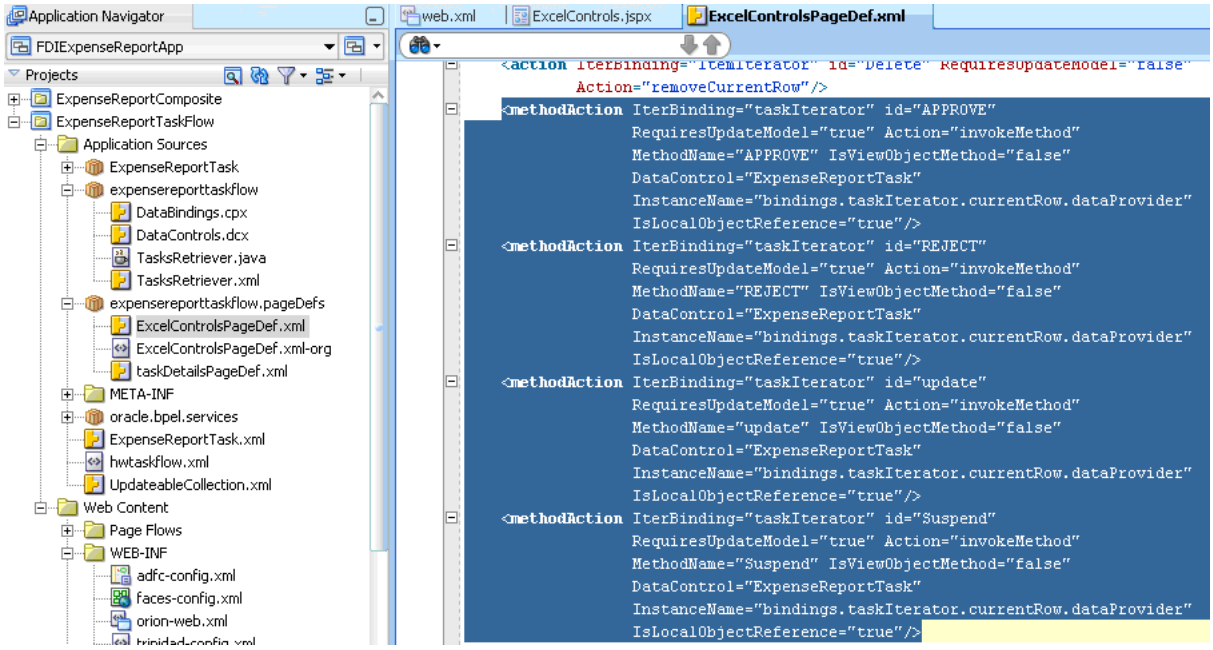
5. Drag and drop the task node from the Data Controls palette to `ExcelControls.jspx`. Select **Human Task**, then select **Complete task with payload**. Figure 32–8 illustrates the sequence of menus you use. Click **OK** on dialogs that pop up.

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.

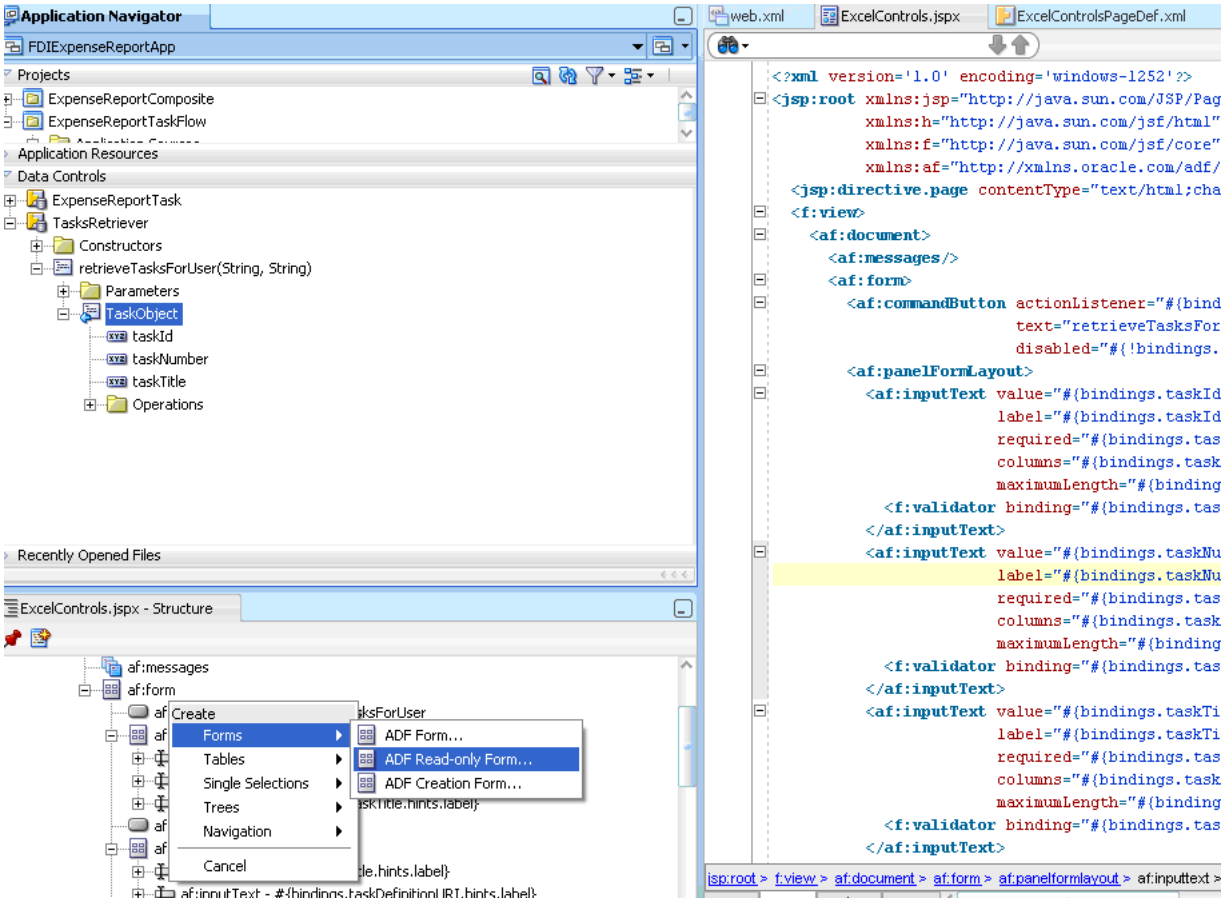
Figure 32–9 Oracle JDeveloper: Configuring the Page Definition File



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.

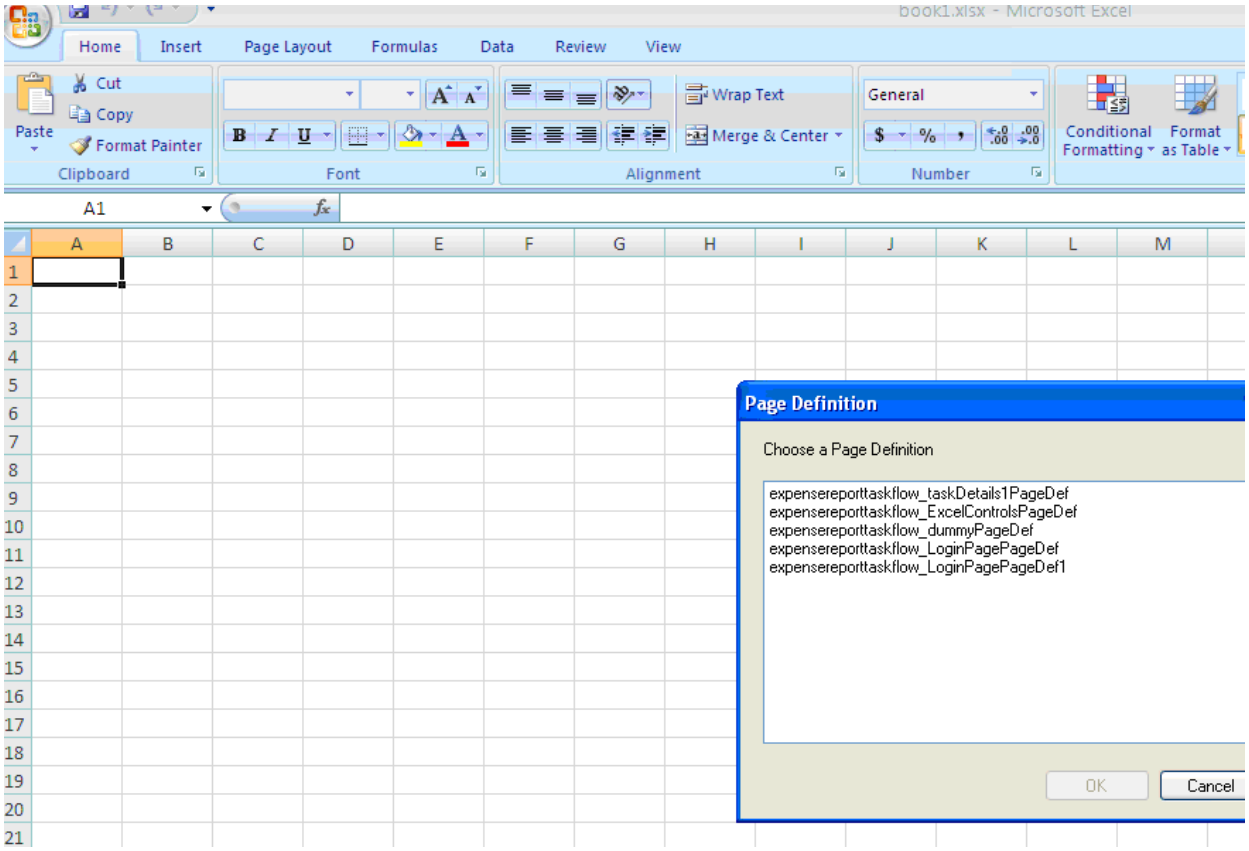
32.2.3.4 Task 4: Prepare the Excel Workbook

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

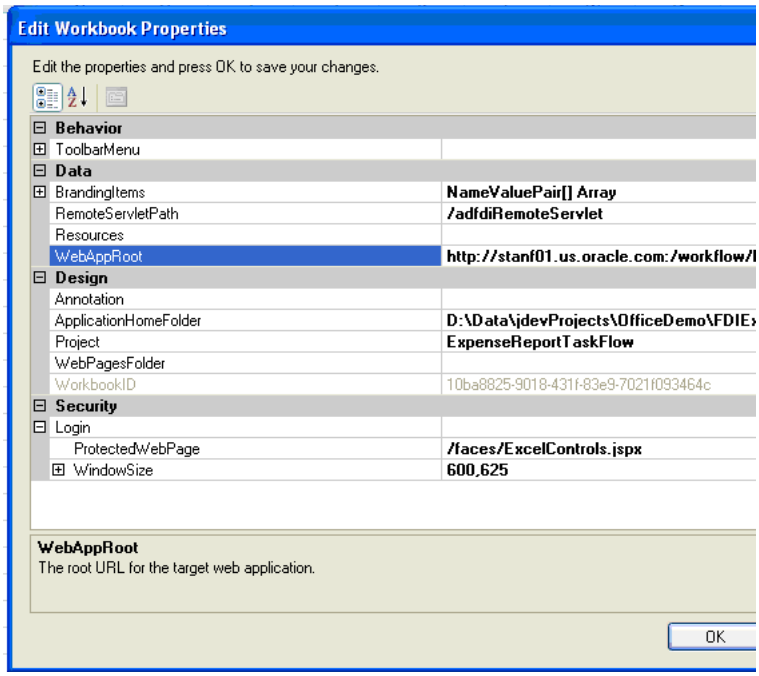


7. In the Document Actions pane, select **Workbook Properties**.
8. Specify ProtectedWebPage: `http://`*`application_server:port`*`//workflow/`*`application_name`*`/faces/app/logon`. (Note that this URL is protected and triggers form authentication. See Section 32.2.3.2, "Task 2: Set up Authentication.")

   Specify WebAppRoot: `http://`*`application_server:port`*`//workflow/`*`application_name`*. Click **OK**.

   Figure 32–12 provides details.

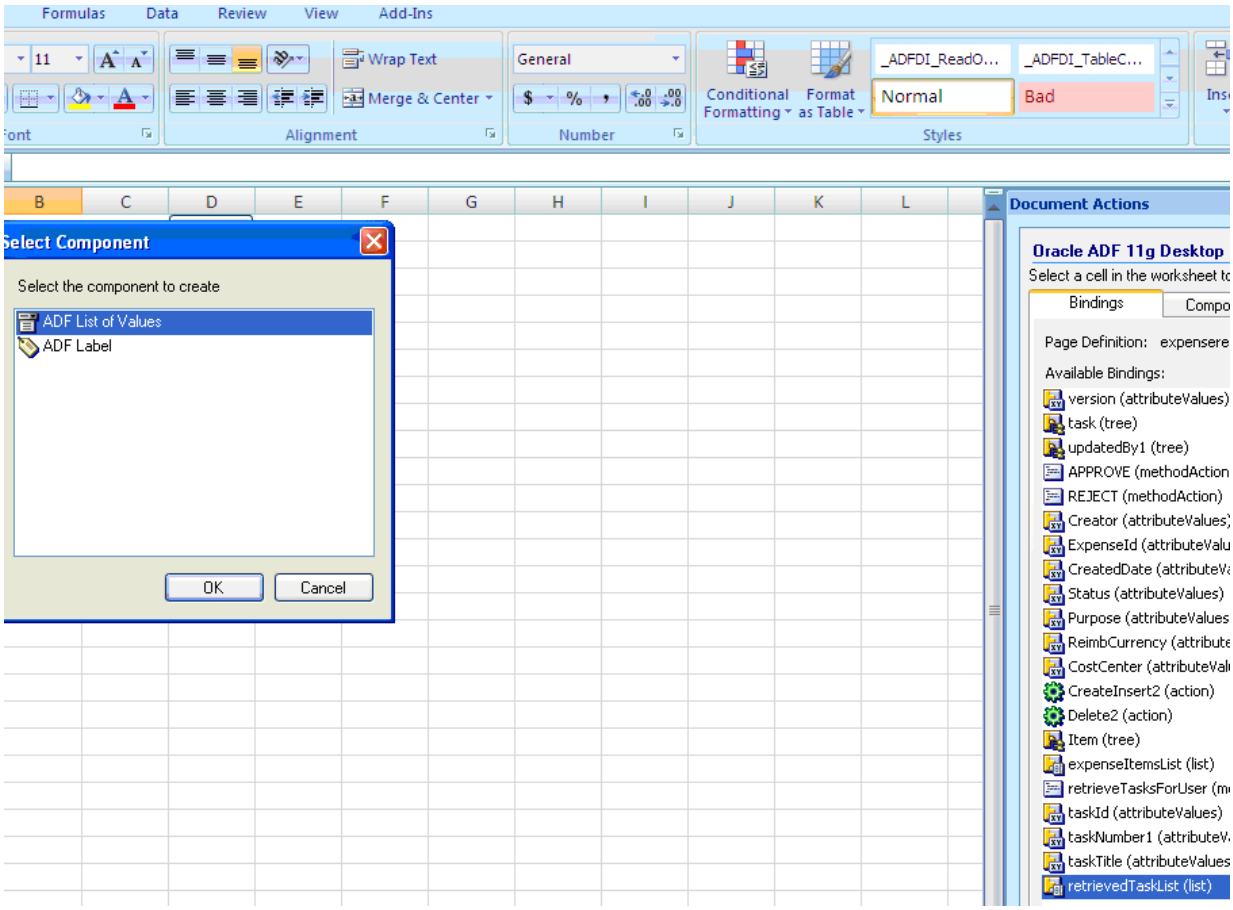
Figure 32–12 Excel: Setting WebAppRoot



For more information, see *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

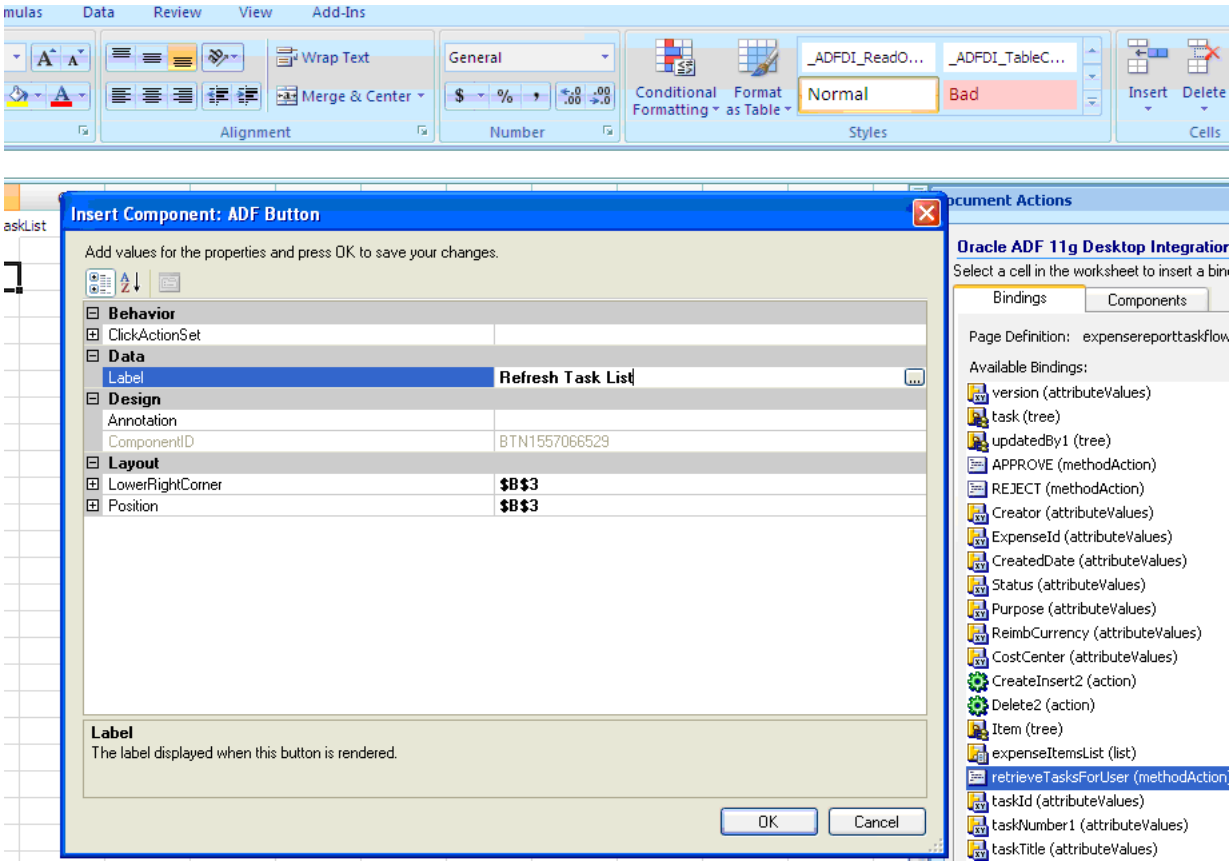
9. From the Document Actions pane, insert ADF Bindings to create the corresponding fields in the Excel workbook. For further details on specific components, see *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*. For instance, insert binding **retrievedTaskList** to create a list of values. Figure 32–13 provides details.

Figure 32–13 Excel: Creating s 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

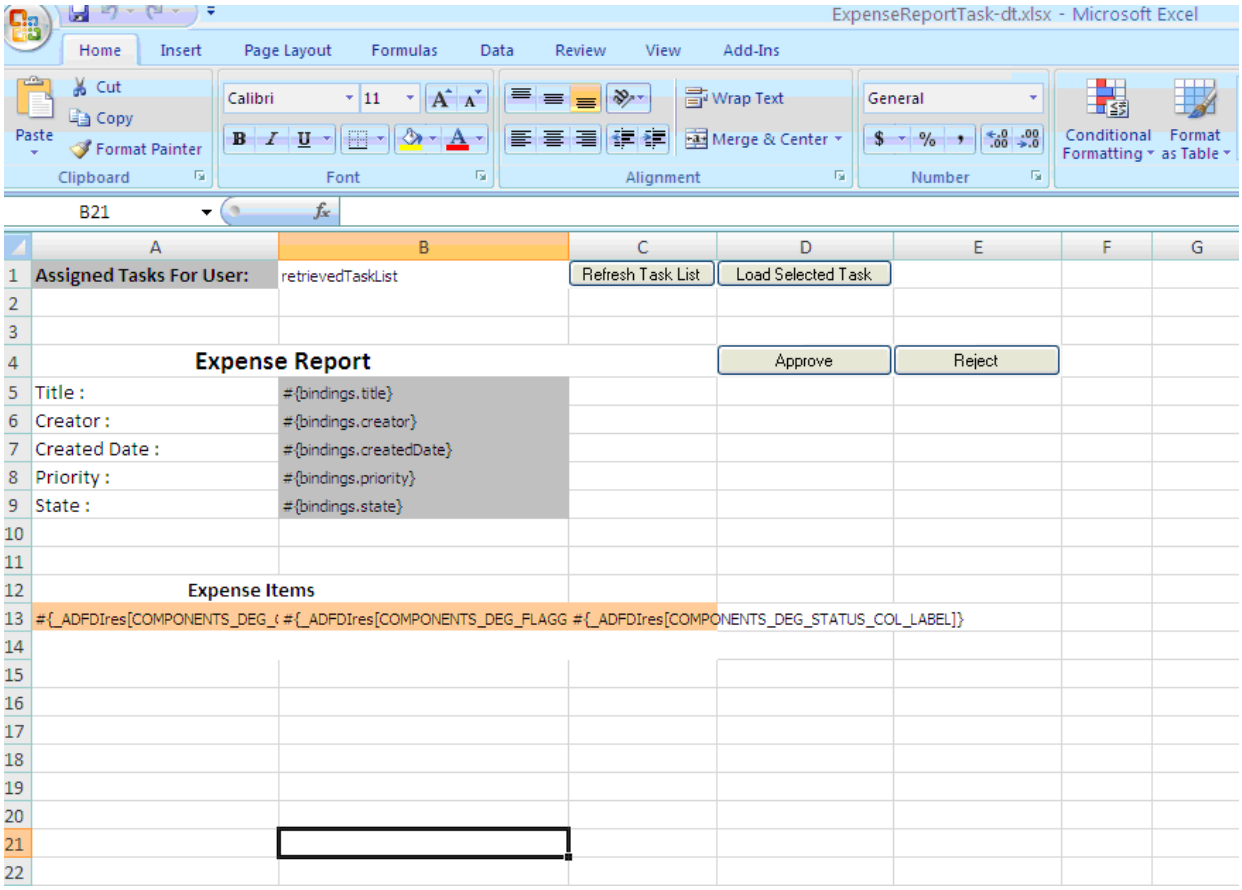


11. Insert a tree binding to create an ADF Table component. A Table component is an updatable table of records in Excel. For instance, the list binding expenseItemsList is a candidate for a Table component.

    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



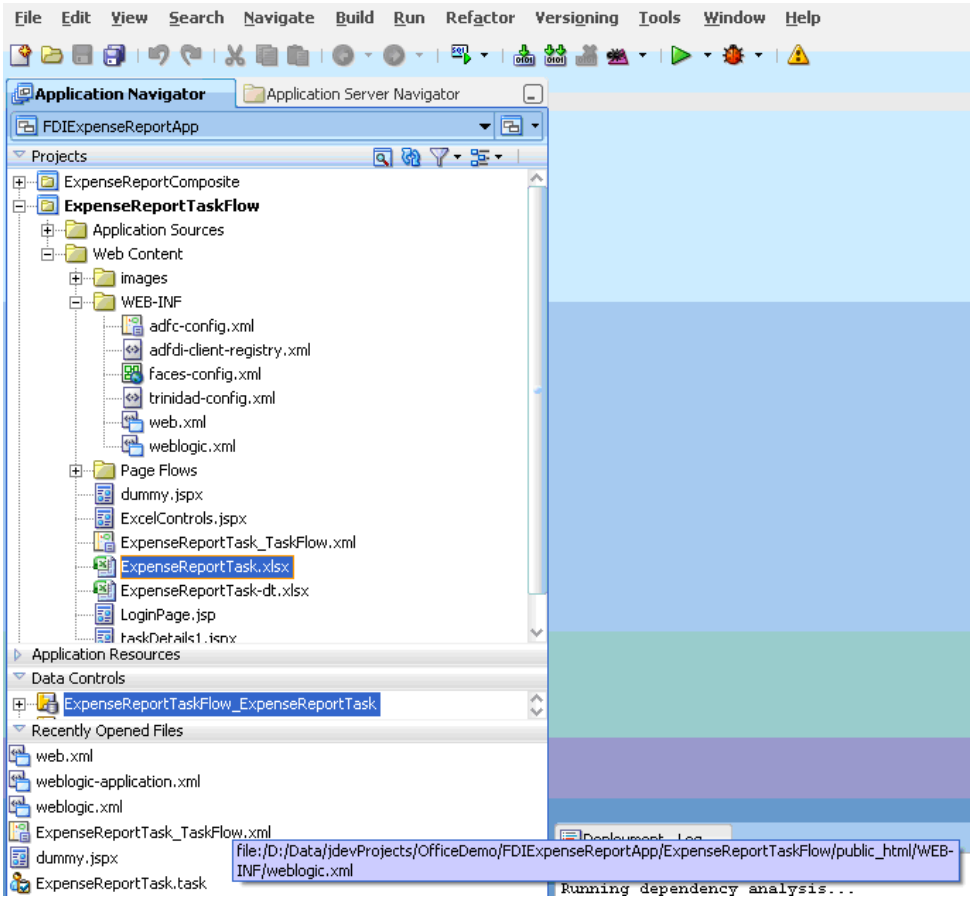
12. Publish the workbook by following these steps:

    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.

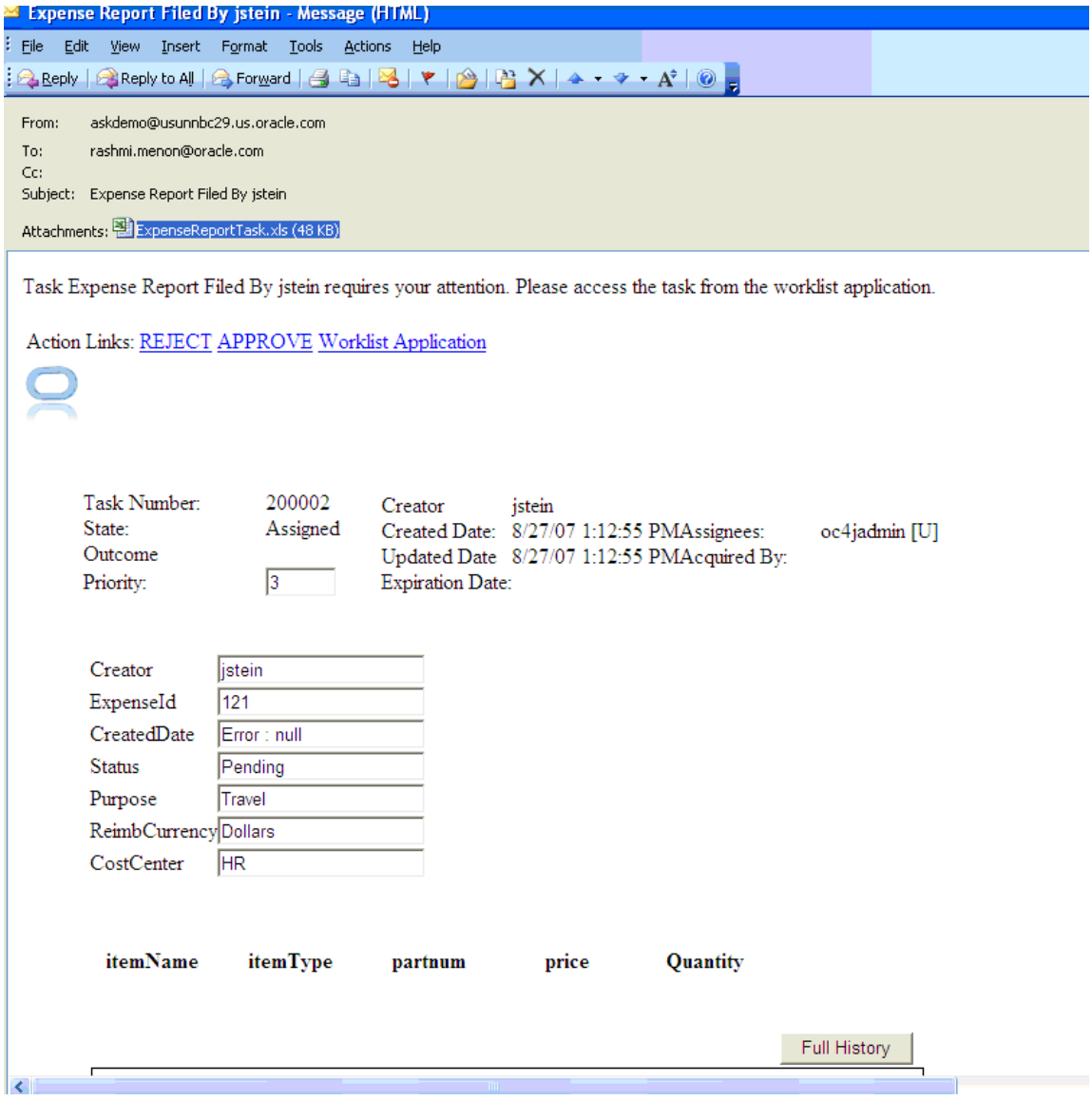
At this point, the ADF task flow is successfully deployed.

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.

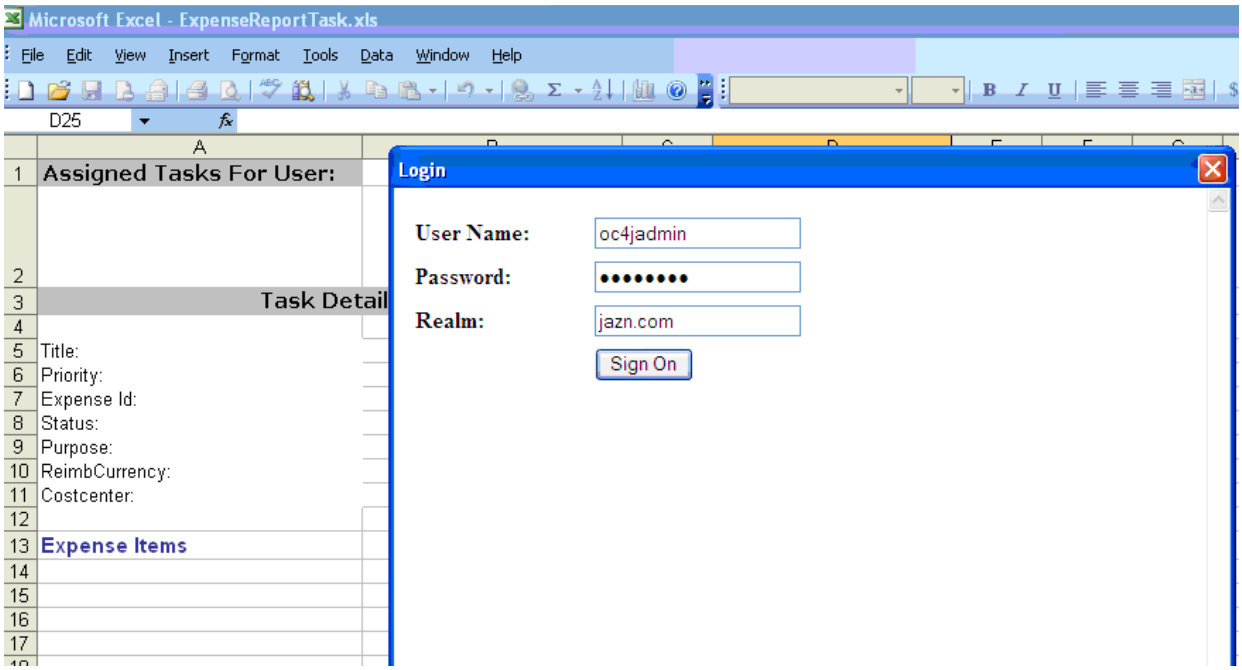
Figure 32–17 Excel Workbook Attached to an Email



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.

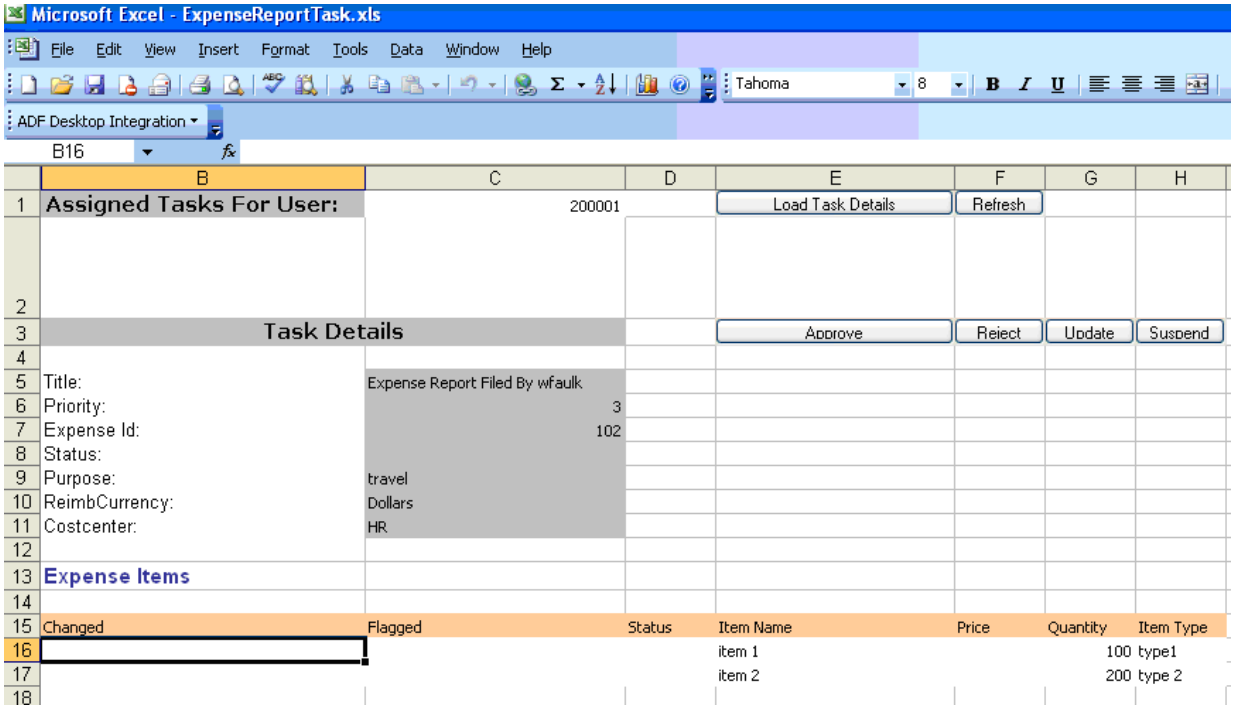
2. Open the Excel workbook. You are directed to a login page (This is `LoginPage.jsp` from Section 32.1.2, "How to Create a Dummy JSF Page.") Enter your security credentials. Figure 32–18 provides details.

Figure 32–18 Desktop-Integrated Excel Workbook: Login Page



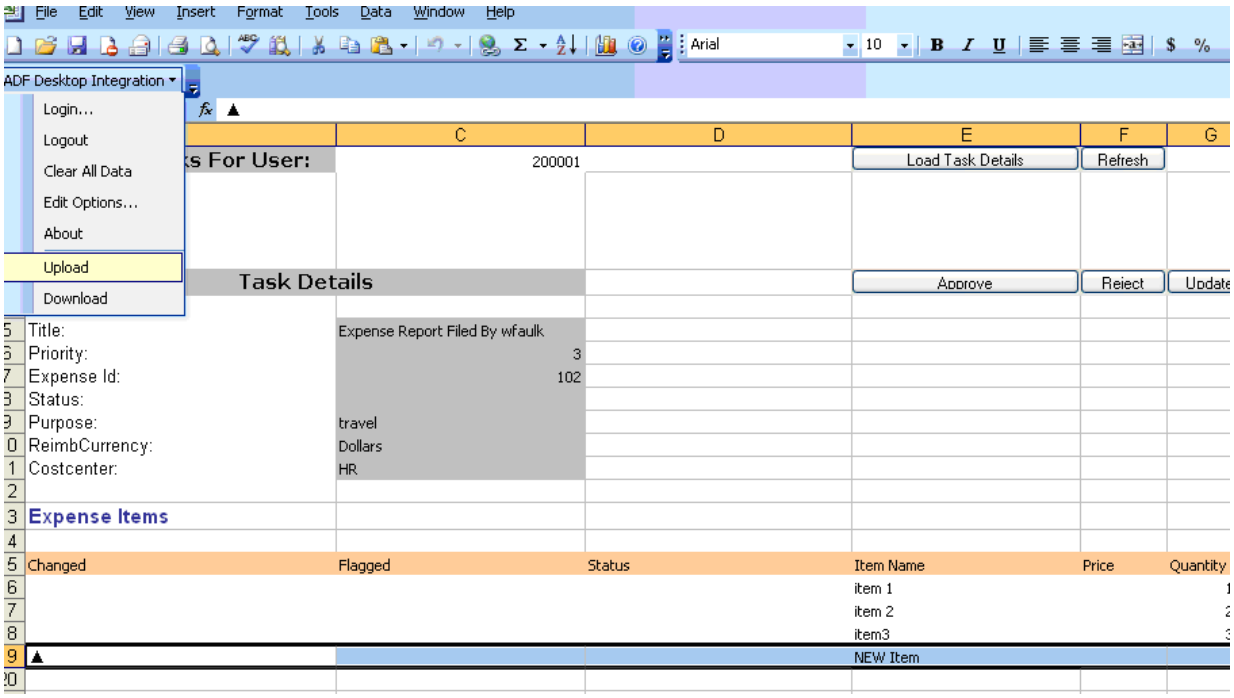
3. Examine the workbook to verify the following:
   - All the assigned tasks for the logged-in user are retrieved in the Excel workbook. Figure 32–19 provides details.

Figure 32–19 ADF Desktop-Integrated Excel Workbook with Assigned Tasks



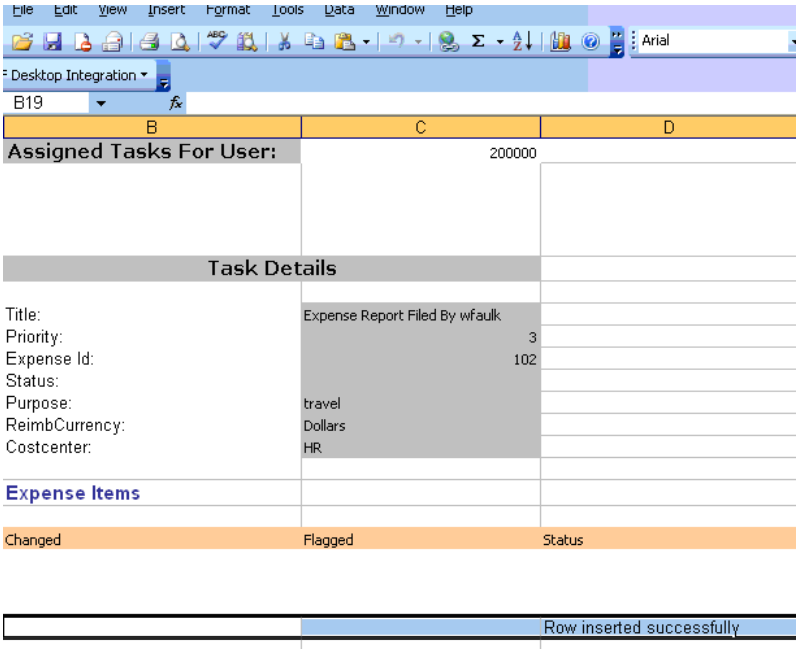
- You can navigate to the needed task from the list of assigned tasks and update it as required. For instance, as illustrated in Figure 32–20, in the Expense Report application, you can upload new expense items.

Figure 32–20 ADF Desktop-Integrated Excel Workbook Uploading New Items



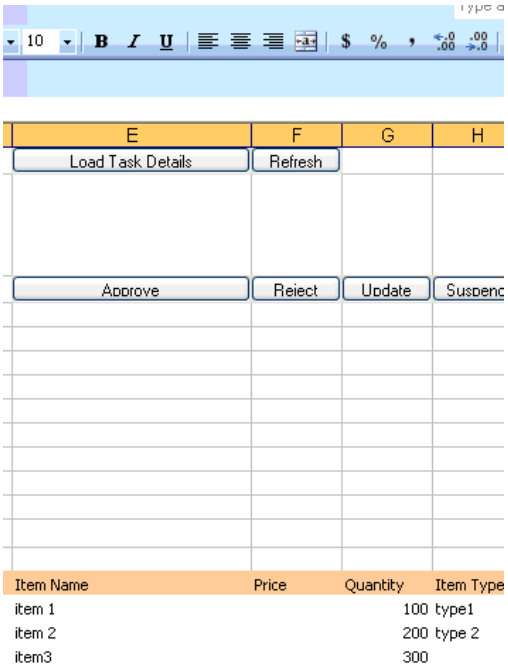
- The Status column in the workbook indicates if the upload was successful. Figure 32–21 provides details.

Figure 32–21 ADF Desktop-Integrated Excel Workbook



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



33

Configuring Task List Portlets

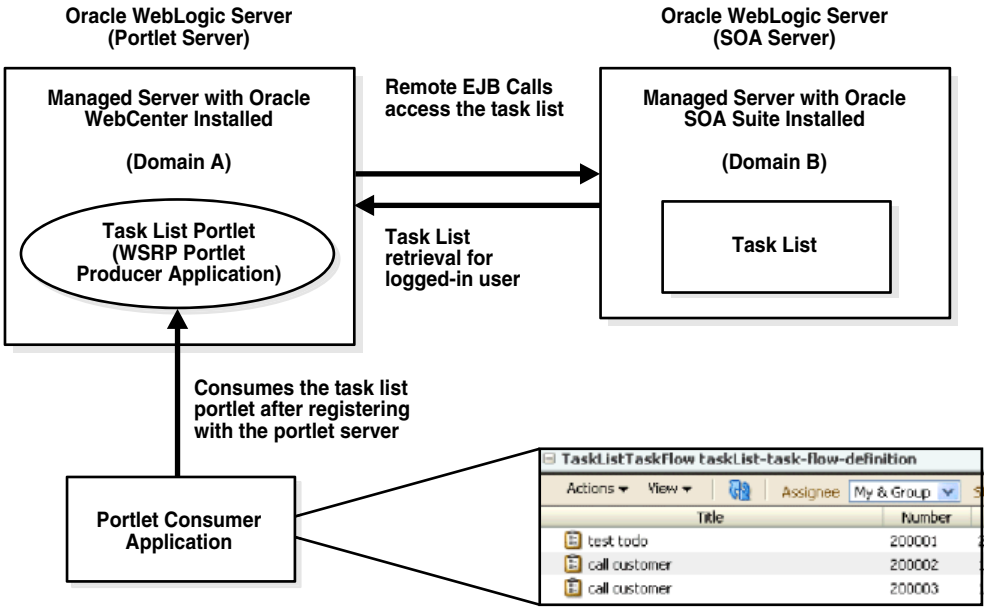
This chapter describes how to configure the task list portlets. This action enables you to review and act upon worklist tasks from an Oracle WebCenter portlet.

This chapter includes the following sections:

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 `weblogic.jndi.WLInitialContextFactory`.
Provider URL	Enter `t3://`*soa_hostname*`:`*soa_port*`/soa-infra`. **Note:** Replace *soa_hostname* and *soa_port* with the hostname and port for the remote Oracle WebLogic Server SOA server that includes the task list to retrieve.
User	Enter `weblogic`.
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 `RuntimeConfigService`.
Local JNDI Name	Enter `RuntimeConfigService`.
Remote JNDI Name	Enter `RuntimeConfigService`.

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	`ejb/bpel/services/workflow/TaskServiceBean`
Second time	`ejb/bpel/services/workflow/TaskMetadataServiceBean`
Third time	`TaskReportServiceBean`
Fourth time	`TaskEvidenceServiceBean`

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.

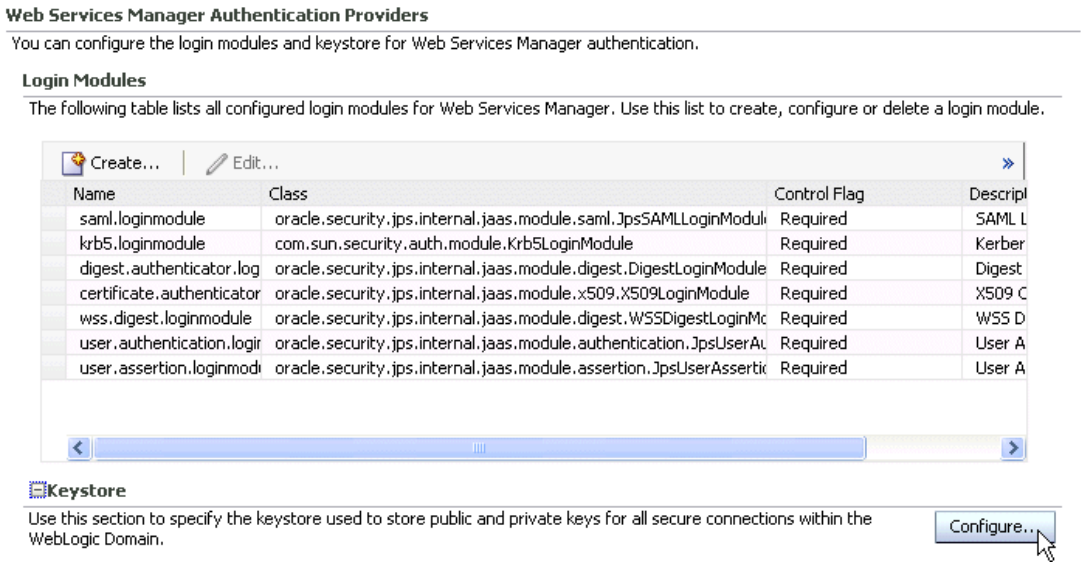
   a. In the navigator on the left side, select **Farm_*base_domain*** > **WebLogic Domain**.

      where ***base_domain*** is the domain name for this example.

   b. Right-click ***base_domain*** and select **Security** > **Security Provider Configuration**.

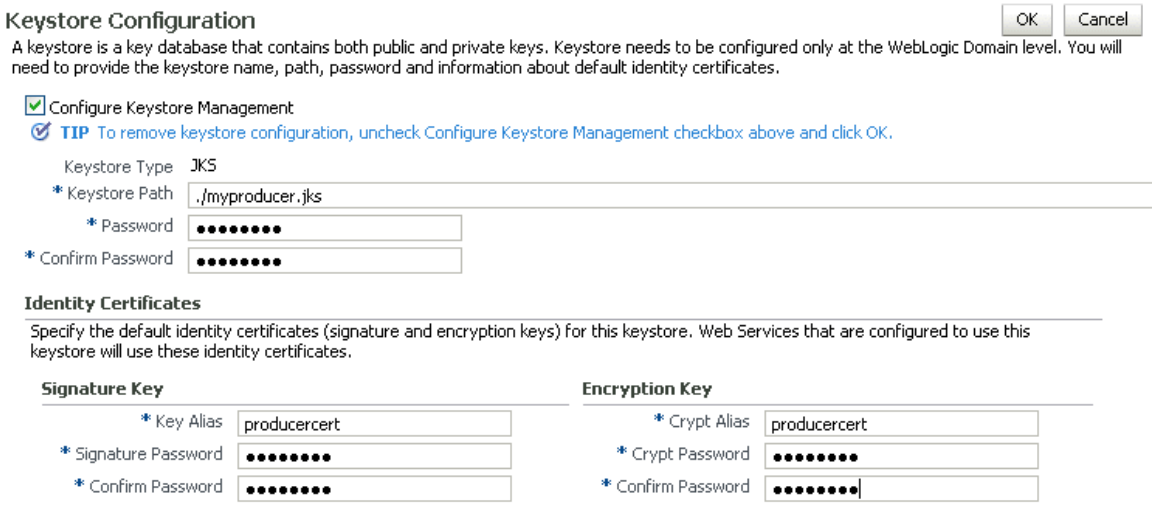
   c. Access the **Keystore** section at the bottom of the provider configuration page and click **Configure**, as shown in Figure 33–2.

Figure 33–2 Keystore Section



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



e. When complete, click **OK**.

f. Restart the managed portlet server and the administration server for the managed portlet server.

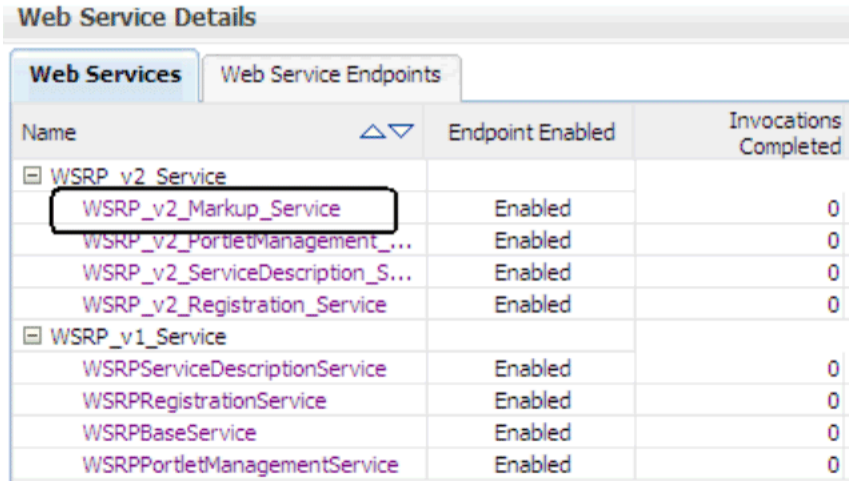
33.2.5 How to Specify the Inbound Security Policy

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:

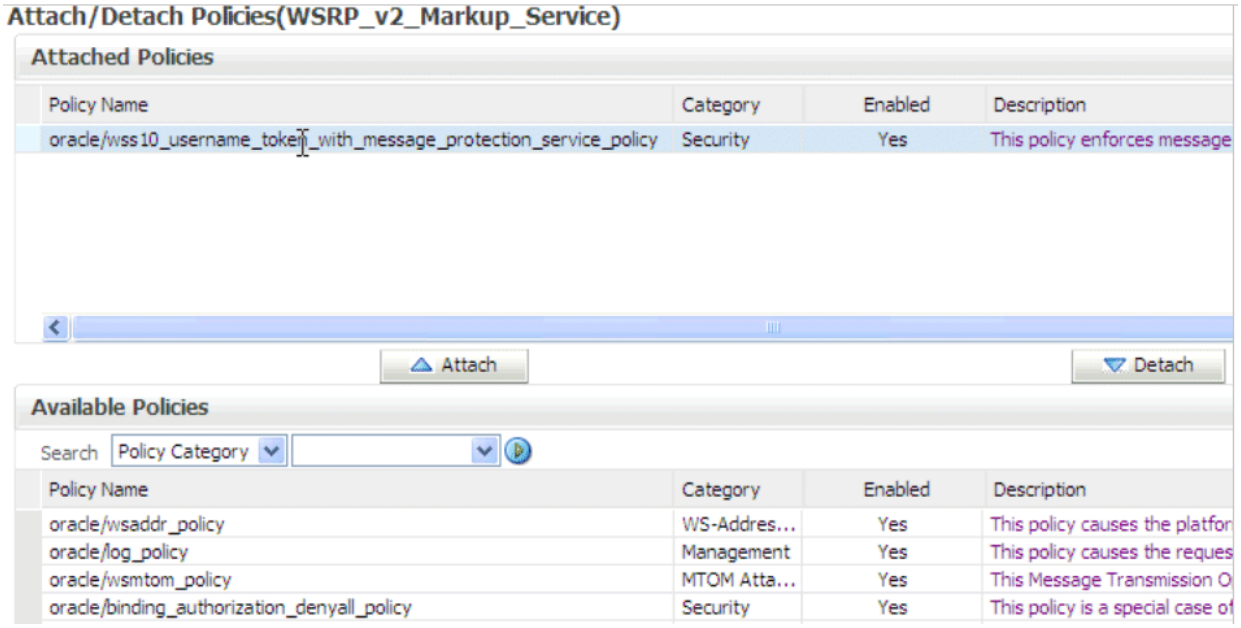
1. In Oracle Enterprise Manager Fusion Middleware Control under **Application Deployments**, navigate to the portlet producer application node.
2. Click **Application Deployments** > **TaskListTaskFlow (WLS_Portlet)**.
3. Select **menu** > **Application Deployments** > **Web Services**.
4. Select the markup port from the page that is displayed, as shown in Figure 33–4.

Figure 33–4 Markup Port Selection



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



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.

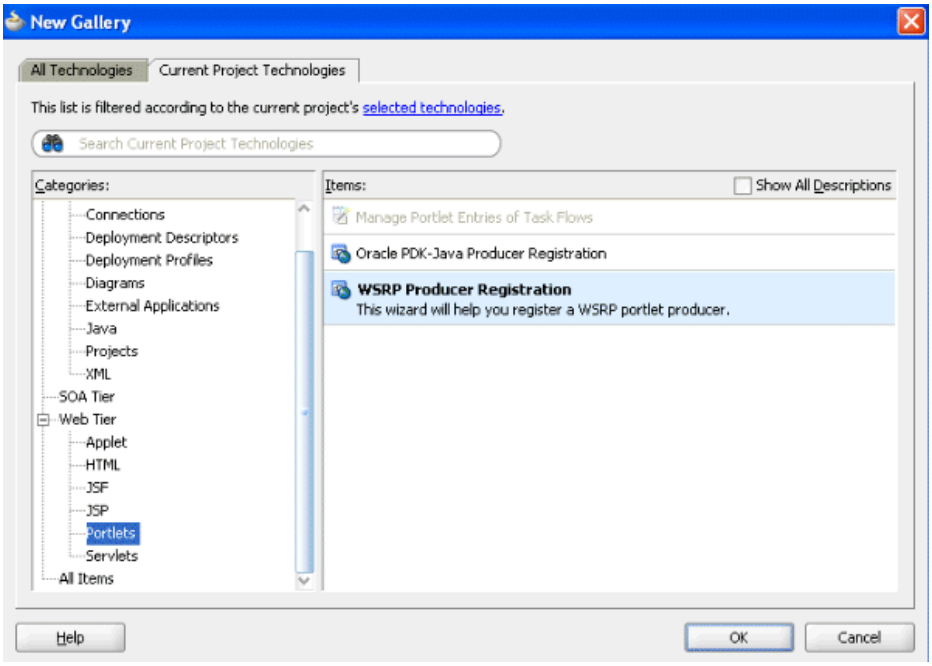
33.3.1 How To Create a Portlet Consumer Application for Embedding the Task List Portlet

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**.

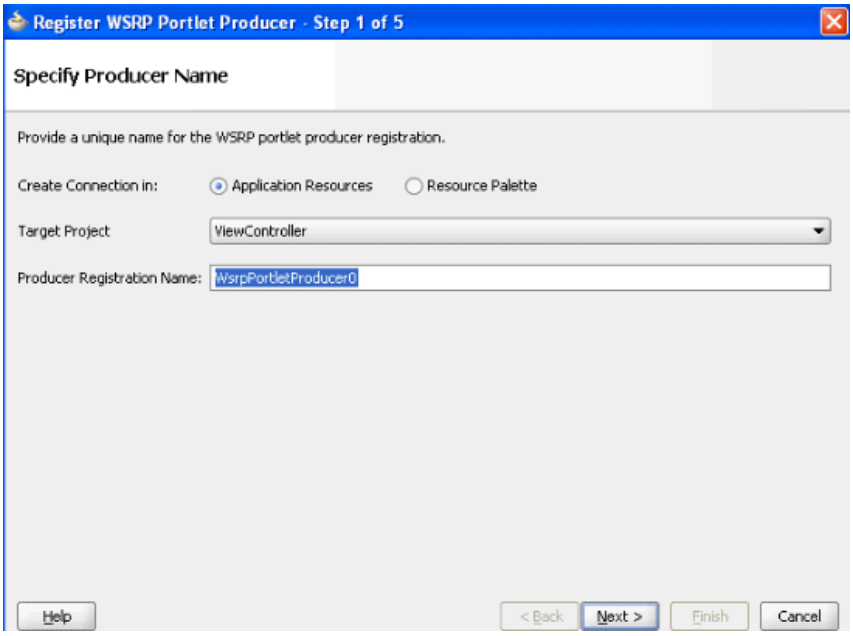
A Register WSRP Portlet Producer wizard is displayed.

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:

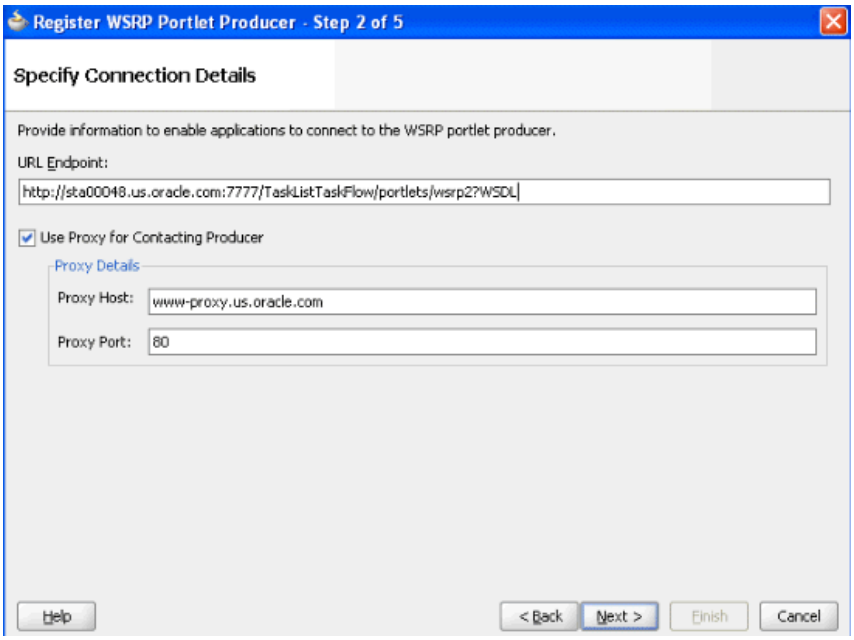
`http://server:port/TaskListTaskFlow/portlets/wsrp2?WSDL`

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.

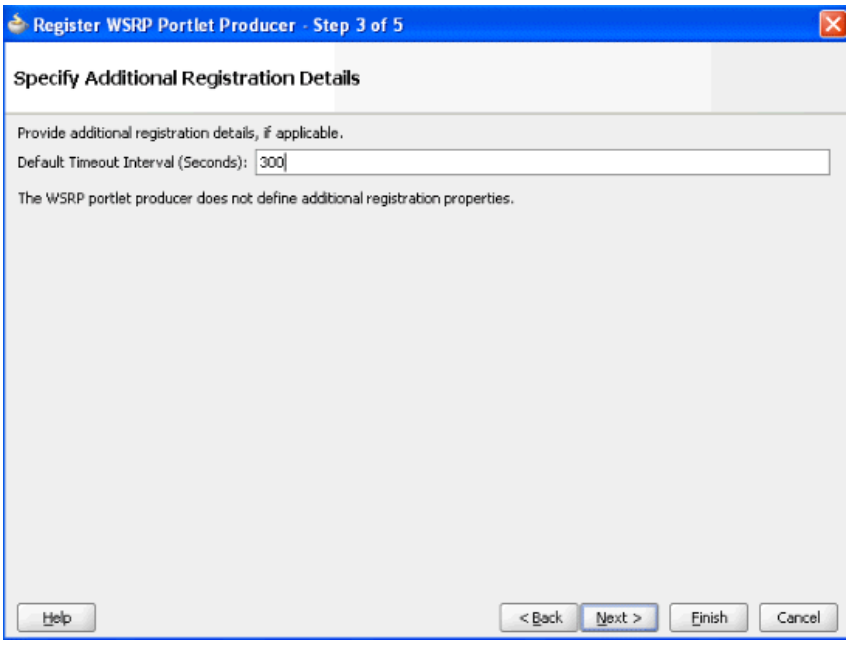
Figure 33–8 URL Endpoint



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

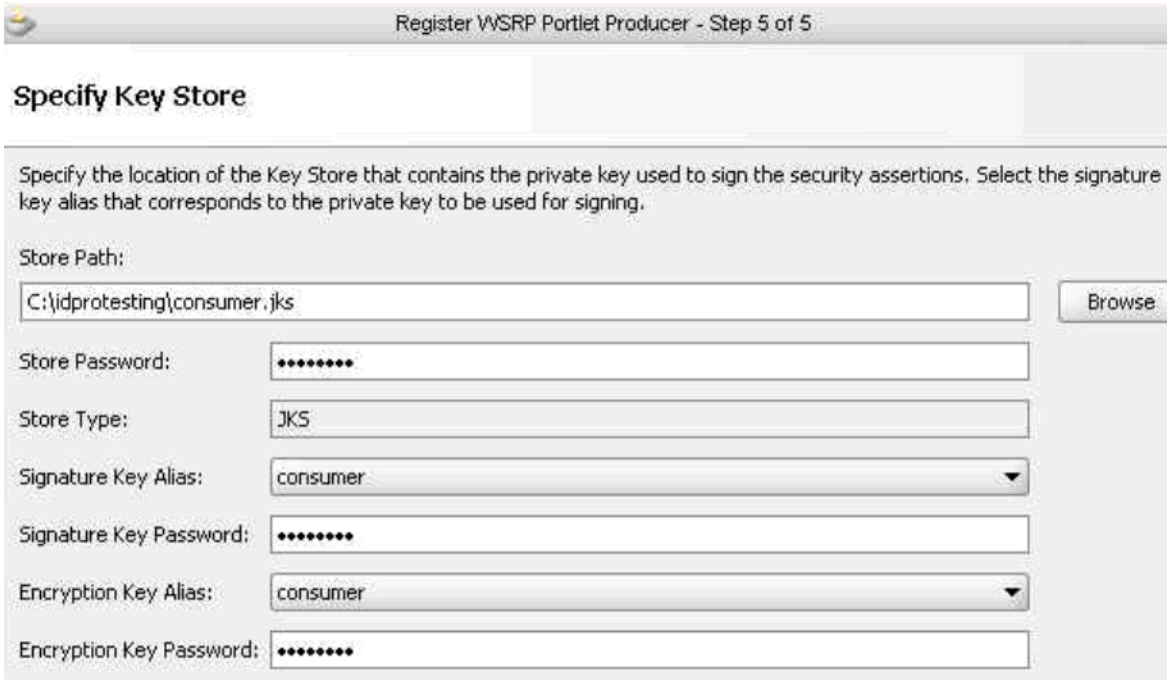
q. Copy `consumer.jks` to your local directory.

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

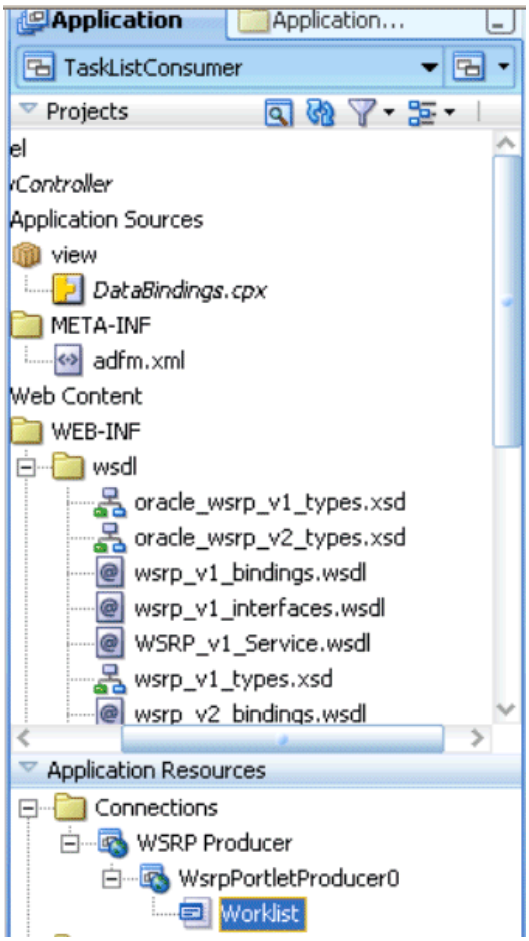


t. Click **Finish**.

The registered portlets appear under **Application Resources**.

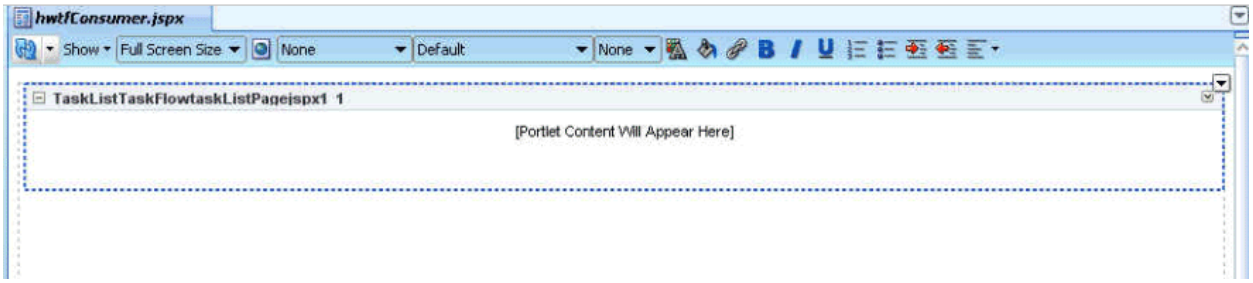
u. Select the token profile based on the requirements of your application, as shown in Figure 33–12.

Figure 33–12 Token Profile Selection



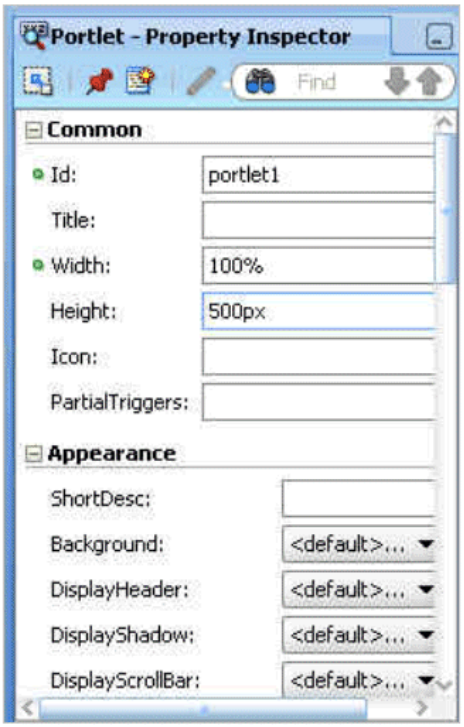
v. Drag the task list portlet named **Worklist** onto the JSPX page `consumer.jspx`, as shown in Figure 33–13.

Figure 33–13 consumer.jspx



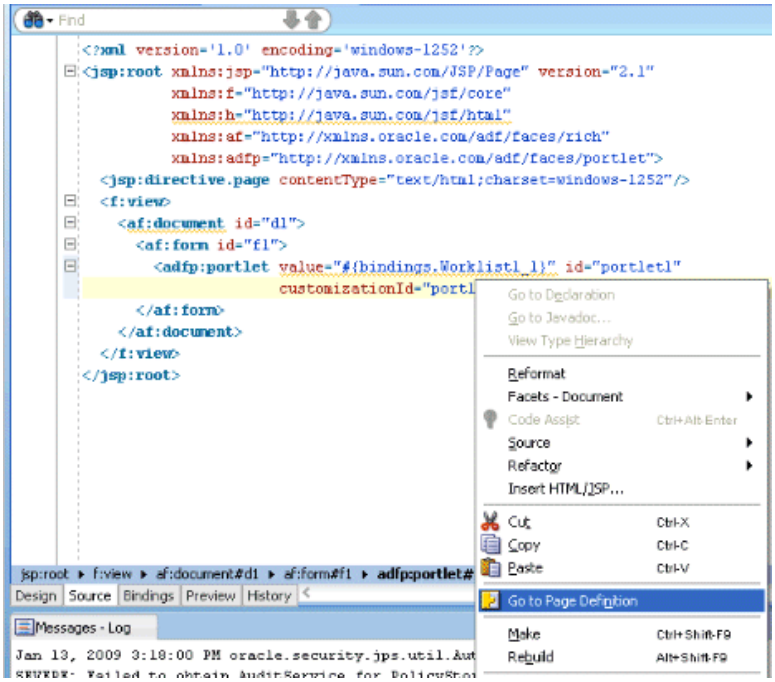
w. Specify the height and width for the task list portlet suitable for your page, as shown in Figure 33–14. This dialog typically appears at the bottom right when you select the portlet component that is dragged onto the page. If this dialog does not appear, select **Property Inspector** from the **View** 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.	`<variable Name="Worklist1_1_soaURL" Type="java.lang.Object" DefaultValue="${'http://soa_host:soa_port')"/>`
`refreshURL` The complete URL of the page, including the task list portlet.	`<variable Name="Worklist1_1_refreshURL" Type="java.lang.Object" DefaultValue="${'http://soa_host:soa_port/HWTFConsumer/faces)`

Figure 33–16 provides details.

Figure 33–16 consumerPageDef.xml

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: `http://soa_host:soa_port`	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 `createdDate`. 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 ACQUIREDBY_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 MOREINFOROLE_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 = "updatedDate";
 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 ISTESTTASK_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 ACTIVTYNAME_COLUMN = "activityName";
 public static final String ACTIVTYID_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,
 // colunm can be used for sorting on client side by FederatedTaskQuerySerive 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 FORMATTRIBUTE1_COLUMN = "formAttribute1";
 public static final String FORMATTRIBUTE2_COLUMN = "formAttribute2";
 public static final String FORMATTRIBUTE3_COLUMN = "formAttribute3";
 public static final String FORMATTRIBUTE4_COLUMN = "formAttribute4";
 public static final String FORMATTRIBUTE5_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 PROTECTEDFORMATTRIBUTE1_COLUMN  =
"protectedFormAttribute1";
 public static final String PROTECTEDFORMATTRIBUTE2_COLUMN  =
"protectedFormAttribute2";
 public static final String PROTECTEDFORMATTRIBUTE3_COLUMN  =
"protectedFormAttribute3";
 public static final String PROTECTEDFORMATTRIBUTE4_COLUMN  =
"protectedFormAttribute4";
 public static final String PROTECTEDFORMATTRIBUTE5_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 FORMATTRIBUTE6_COLUMN  = "formAttribute6";
 public static final String FORMATTRIBUTE7_COLUMN  = "formAttribute7";
 public static final String FORMATTRIBUTE8_COLUMN  = "formAttribute8";
 public static final String FORMATTRIBUTE9_COLUMN  = "formAttribute9";
 public static final String FORMATTRIBUTE10_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 PROTECTEDFORMATTRIBUTE6_COLUMN  =
"protectedFormAttribute6";
 public static final String PROTECTEDFORMATTRIBUTE7_COLUMN  =
"protectedFormAttribute7";
 public static final String PROTECTEDFORMATTRIBUTE8_COLUMN  =
"protectedFormAttribute8";
 public static final String PROTECTEDFORMATTRIBUTE9_COLUMN  =
"protectedFormAttribute9";
 public static final String PROTECTEDFORMATTRIBUTE10_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 = ",";

}
```

Part VI

Using Binding Components

This section describes how to use binding components.

This part contains the following chapters:

34

Getting Started with Binding Components

This chapter provides a high-level overview of supported binding component types and technologies that you can integrate in a SOA composite application. This chapter also provides references to documentation that more fully describes these technologies.

This chapter includes the following sections:

- Section 34.1, "Introduction to Binding Components"
- Section 34.2, "Introduction to Integrating a Binding Component in a SOA Composite Application"

34.1 Introduction to Binding Components

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

- Services

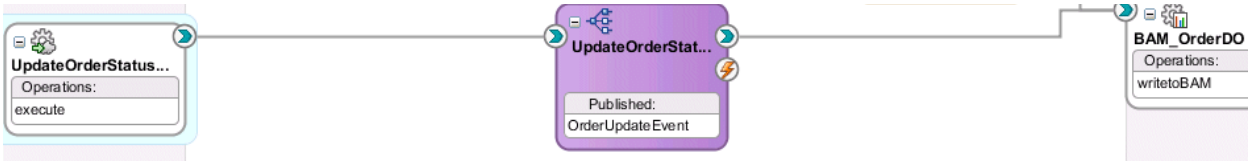
  Provide the outside world with an entry point to the SOA composite application. The WSDL file of the service advertises its capabilities to external applications. These capabilities are used for contacting the SOA composite application components. The binding connectivity of the service describes the 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.

Figure 34–1 shows the **OrderBookingComposite** project in the Fusion Order Demo in which a service (**UpdateOrderStatus**) in the **Exposed Services** swimlane provides the entry point to the composite and a reference (**BAM_OrderDO**) in the **External References** swimlane enables information to be sent to an Oracle Business Activity Monitoring (BAM) Server in the outside world.

Figure 34–1 Service and Reference Binding Components



Binding components enable you to integrate the following types of technologies with SOA composite applications:

- Web services
- HTTP binding
- JCA adapters
- Oracle Business Activity Monitoring (BAM)
- Oracle B2B
- ADF-BC services
- EJB services
- Direct binding services

These technologies are described in the following sections.

34.1.1 Web Services

This service enables you to integrate applications with a standards-based web service using SOAP over HTTP. Web services are described in the WSDL file.

Dragging a web service into a swimlane of the SOA Composite Editor invokes the Create Web Service dialog for specifying configuration properties.

For more information about web services, see Section 2.3.2, "How to Add a WSDL for a Web Service."

For information about adding Message Transmission Optimization Mechanism (MTOM) attachments to web services, see Section 42.1.1.3, "Adding MTOM Attachments to Web Services."

34.1.1.1 WS-AtomicTransaction Support

The Create Web Service dialog also enables you to configure support for WS-Coordination and WS-AtomicTransaction (WS-AT) transactions. WS-AT provides transaction interoperability between Oracle WebLogic Server and other vendors' transaction services. Interoperability is provided at two levels:

- Exporting transactions from the local Java Transaction API (JTA) environment for a web service request.
- Importing transactions from a web service request into the local JTA environment. This allows for distributed transaction processing between multiple nodes in the web services environment.

Figure 34–2 shows the support for WS-AT at the bottom of the Create Web Service dialog.

Figure 34–2 WS-AT Support in Create Web Service Dialog

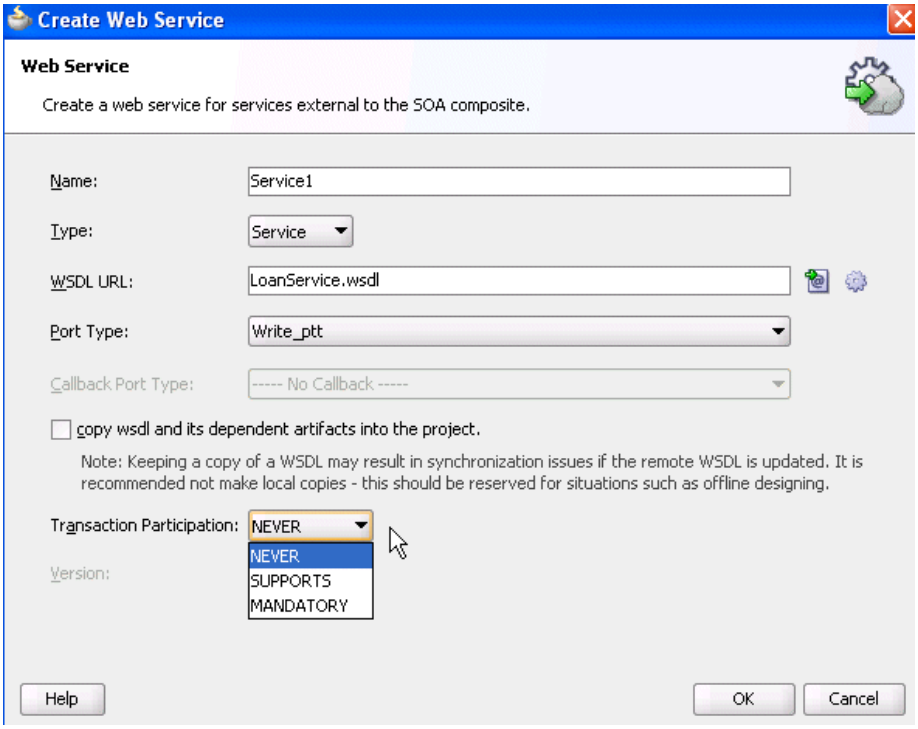


Table 34–1 describes the WS-AT fields. For a description of the remaining fields in the Create Web Service dialog, see Section 2.3.2, "How to Add a WSDL for a Web Service."

Table 34–1 WS-AT Fields of the Create Web Service Dialog

Property	Description
Transaction Participation	Select a value. If you added the web service to the **Exposed Services** swimlane, this action enables external transaction managers to coordinate resources hosted on Oracle WebLogic Server over WS-AT. If you added the web service to the **External References** swimlane, this addition enables Oracle WebLogic Server transactions to coordinate resources hosted in external environments over WS-AT. ▪ **Never** No transaction context is imported (for services) or exported (for references). This is the default value if you add the web service as a service binding component in the **Exposed Services** swimlane. ▪ **Supports** If a transaction exists, a transaction context is imported (for services) or exported (for references). This information is added to the `composite.xml` file. ▪ **Mandatory** A transaction context is imported (for services) or exported (for references). This information is added to the `composite.xml` file. For exports, a web service exception message is thrown if there is no active transaction. For imports, a fault is returned to the client if there is no transaction context in the request. ▪ **WSDLDriven** This property only displays if you add the web service as a reference binding component in the **External References** swimlane. This is the default value.

Table 34–1 (Cont.) WS-AT Fields of the Create Web Service Dialog

Property	Description
Version	Displays the WS-AT supported version (1.0, 1,1, 1,2, or default). By default, this list is only enabled if you select **Supports** or **Mandatory** from the **Transaction Participation** list.

When complete, the `composite.xml` file displays your WS-AT selections, as shown in Example 34–1.

Example 34–1 WS-AT Syntax in composite.xml File

```
  <service name="Service1" ui:wsdlLocation="BPELProcess1.wsdl">
    <interface.wsdl interface="http://xmlns.oracle.com/Application5_
jws/Project1/BPELProcess1#wsdl.interface(BPELProcess1)"
                    callbackInterface="http://xmlns.oracle.com/Application5_
jws/Project1/BPELProcess1#wsdl.interface(BPELProcess1Callback)"/>
    <binding.ws port="http://xmlns.oracle.com/Application5_
jws/Project1/BPELProcess1#wsdl.endpoint(Service1/BPELProcess1_pt)">
      <property name="weblogic.wsee.wsat.transaction.flowOption"
                type="xs:string" many="false">SUPPORTS</property>
      <property name="weblogic.wsee.wsat.transaction.version" type="xs:string"
                many="false">WSAT11</property>
    </binding.ws>
```

If you want to edit your changes, you can right-click the service and select **Edit** or double-click the service in the SOA Composite Editor.

After deployment, you can modify the transaction participation and version values through the System MBean Browser. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

For more information about WS-AT and WS-Coordination, see *Oracle Fusion Middleware Concepts Guide for Oracle Infrastructure Web Services* and the WS-AT and WS-Coordination specifications, which are available at the following URL:

`http://www.oasis-open.org`

34.1.1.1.1 Ensuring Participation of BPEL Processes in WS-AT In addition to setting the WS-AT participation property, if a client calls a web service that is a BPEL process, for that web service to be enlisted in the caller's transaction, the callee BPEL process must have the `transaction` property set in its `composite.xml` file.

```
<property name="bpel.config.transaction">required</property>
```

This setting ensures that, if an error occurs (such as a database adapter invocation failing due to an integrity constraint violation), a transaction rollback is successfully completed.

For more information about setting the `transaction` property, see Section C.1.1, "How to Define Deployment Descriptor Properties" and Section 12.1.1, "Oracle BPEL Process Manager Transaction Semantics."

34.1.1.1.2 WS-AT Transactions are Not Supported When Optimization is Enabled You can configure a web service binding component as either a service or reference to support WS-AT transactions from the **Transaction Participation** dropdown list of the Create Web Service dialog. WS-AT transactions are supported in composite-to-web service environments, or vice-versa, with the `oracle.webservices.local.optimization` property set to `false`.

WS-AT transactions are not supported in composite-to-composite calls, even with the `oracle.webservices.local.optimization` property set to `false`.

For more information about the `oracle.webservices.local.optimization` property, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

34.1.2 HTTP Binding Service

The HTTP binding service enables you to integrate SOA composite applications with HTTP binding.

You drag the **HTTP Binding** service from the Component Palette into a swimlane of the SOA Composite Editor to invoke the HTTP Binding Wizard. This addition enables you to configure HTTP binding as follows:

- As a service binding component in the **Exposed Services** swimlane to invoke SOA composite applications through HTTP POST and GET operations
- As a reference binding component in the **External References** swimlane to invoke HTTP endpoints through HTTP POST and GET operations

---

Note: Note the following details about using HTTP binding in a SOA composite application.

- An outbound HTTP binding reference supports only XML as a response from an external HTTP endpoint. The response should contain the correct XML part name according to outbound expectations.
- You cannot change the **httpBinding** property for the HTTP binding component during runtime in Oracle Enterprise Manager Fusion Middleware Control.

34.1.2.1 Supported Interactions

Table 34–2 shows the supported verbs, payloads, and operations for the inbound and outbound directions.

Table 34–2 Supported Verbs, Payloads, and Operations

Direction	Verb	Payload Type	Operation	Supported?
Inbound	GET	URL-encoded	One-way	Yes
Inbound	GET	URL-encoded	Request-response	Yes
Inbound	GET	XML	One-way	No
Inbound	GET	XML	Request-response	No
Inbound	POST	URL-encoded	One-way	Yes
Inbound	POST	URL-encoded	Request-response	Yes
Inbound	POST	XML	One-way	Yes
Inbound	POST	XML	Request-response	Yes
Outbound	GET	URL-encoded	One-way	No
Outbound	GET	URL-encoded	Request-response	Yes
Outbound	GET	XML	One-way	No

Table 34–2 (Cont.) Supported Verbs, Payloads, and Operations

Direction	Verb	Payload Type	Operation	Supported?
Outbound	GET	XML	Request-response	Yes
Outbound	POST	URL-encoded	One-way	No
Outbound	POST	URL-encoded	Request-response	Yes
Outbound	POST	XML	One-way	No
Outbound	POST	XML	Request-response	Yes

Table 34–3 shows the supported XSD types for the inbound and outbound directions.

Table 34–3 Supported XSDs

Direction	XSD Type	Supported?
Inbound	Simple	Yes
Inbound	Complex	No
Inbound	Native	No
Outbound	Simple	Yes
Outbound	Complex	No
Outbound	Native	No

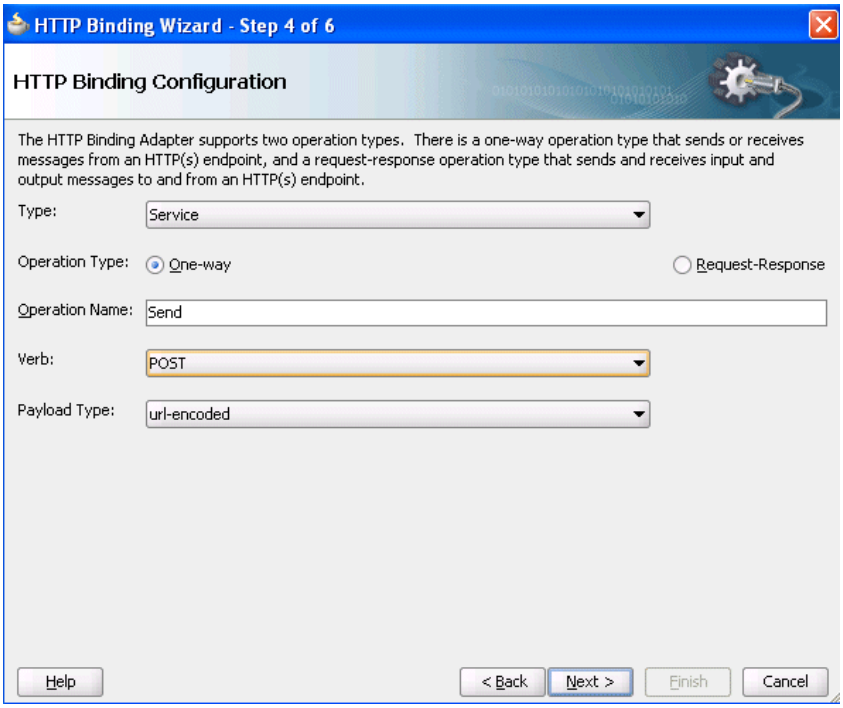
The following HTTP headers are not supported in either the inbound or outbound direction (that is, you cannot access HTTP headers in the composite and set them in the composite):

- `User-agent`
- `Content-type`
- `Content-length`
- `Server`
- `Server-port`
- `Referrer`
- `Authorization`
- `MIME-Version`
- `Location`

34.1.2.2 How to Configure the HTTP Binding Service

You invoke the HTTP Binding Wizard to configure HTTP binding by dragging the **HTTP Binding** icon from the Component Palette. The HTTP Binding Component page of the wizard enables you to specify the operation type, verb, and payload type. Figure 34–3 provides details.

Figure 34–3 Create HTTP Binding Wizard - HTTP Binding Configuration Page



This page of the wizard enables you to select the following operation types for inbound HTTP binding:

- A one-way operation that sends or receives messages to or from an HTTP endpoint
- A synchronous request-response operation that sends and receives input and output messages to and from an HTTP endpoint

For HTTP POST request methods, you can select a payload type of either URL-encoded (ampersand-separated name-value pairs) or XML.

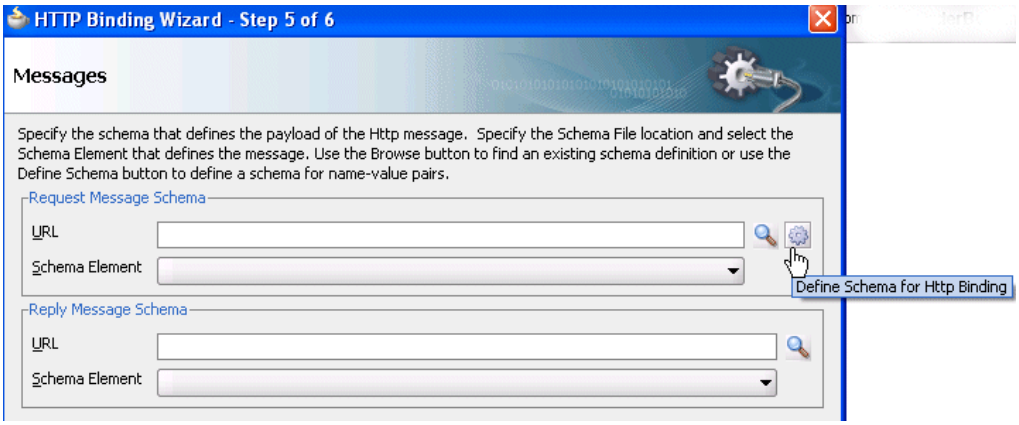
For HTTP GET request methods, the payload type is URL-encoded.

For HTTP GET or POST request methods for reference binding components, you are also prompted to specify the endpoint URL. Support for HTTP authentication and secure socket layer (SSL) is also provided.

> **Note:** Secure HTTP (HTTPS) is supported in both the inbound and outbound directions.

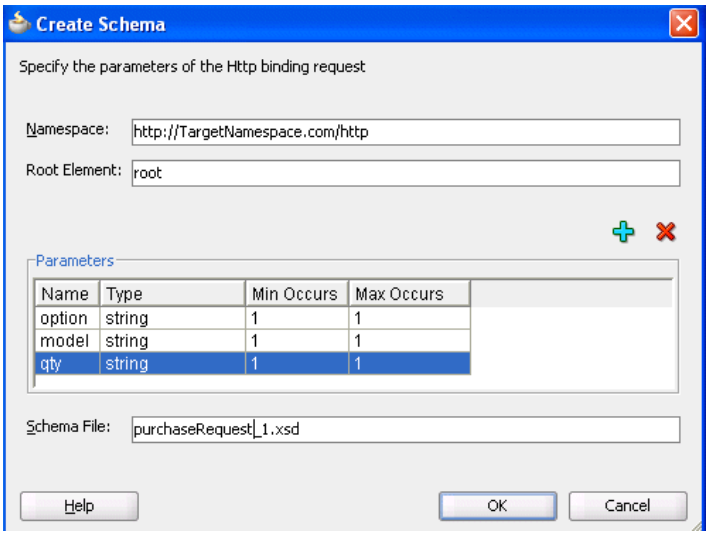
During the configuration process with the HTTP Binding Wizard, you have the option of browsing for an existing request message schema or defining your own schema with the links to the right of the **URL** field on the Messages page. Figure 34–4 provides details.

Figure 34–4 Create HTTP Binding Wizard - Messages Page



If you select to define your own schema, you are prompted to specify the element names, data types, min occurs value, and max occurs value in the Create Schema dialog. Figure 34–5 provides details.

Figure 34–5 Create HTTP Binding Wizard - Create Schema Page



At runtime, the concrete WSDL is generated with an HTTP binding and a SOAP binding; this is because the SOAP endpoint is used to provide HTTP support.

34.1.2.3 How to Enable Basic Authentication

Inbound and outbound HTTP binding supports basic authentication. If you want to enable basic authentication for inbound HTTP binding, you must attach a security policy. Note that inbound HTTP binding can also be used without enabling basic authentication.

To enable basic authentication:

1. Right-click the created HTTP binding service in the **Exposed Services** swimlane and select **Configure WS Policies**.
2. In the Configure SOA WS Policies dialog, click the **Add** icon in the **Security** section.

3. Select the **oracle/wss_http_token_service_policy** policy, and click **OK**.
4. In the Configure SOA WS Policies dialog, click **OK**.

34.1.3 JCA Adapters

JCA adapters enable you to integrate services and references with the following technologies:

- Databases
- File systems
- FTP servers
- Message systems such as Advanced Queueing (AQ) and Java Messaging Systems (JMS)
- IBM WebSphere MQ
- TCP/IP sockets
- Third-party adapters (SAP, PeopleSoft, and others)

Dragging a JCA adapter into a swimlane of the SOA Composite Editor invokes the Adapter Configuration Wizard for specifying configuration properties.

34.1.3.1 AQ Adapter

The AQ adapter enables you to interact with a single consumer or multiconsumer queue.

Oracle Streams AQ provides a flexible mechanism for bidirectional, asynchronous communication between participating applications. Advanced queues are an Oracle database feature, and are therefore scalable and reliable. Multiple queues can also service a single application, partitioning messages in a variety of ways and providing another level of scalability through load balancing.

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

34.1.3.2 Database Adapter

The database adapter enables a BPEL process to communicate with Oracle databases or third-party databases through JDBC.

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

34.1.3.3 File Adapter

The file adapter enables a BPEL process or Oracle Mediator to exchange (read and write) files on local file systems. The file contents can be in both XML and non-XML data formats.

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

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

34.1.3.4 FTP Adapter

The FTP adapter enables a BPEL process or Oracle Mediator to exchange (read and write) files on remote file systems through use of the file transfer protocol (FTP). The file contents can be in both XML and non-XML data formats.

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

34.1.3.5 JMS Adapter

The JMS adapter enables an Oracle BPEL process or Oracle Mediator to interact with a Java Messaging System (JMS).

The JMS architecture uses one client interface to many messaging servers. The JMS model has two messaging domains:

- Point-to-point: Messages are exchanged through a queue and each message is delivered to only one receiver.
- Publish-subscribe: Messages are sent to a topic and can be read by many subscribed clients.

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

34.1.3.6 MQ Adapter

The MQ adapter provides message exchange capabilities between BPEL processes and Oracle Mediator and the WebSphere MQ queuing systems.

The Messaging and Queuing Series (MQ Series) is a set of products and standards developed by IBM. MQ Series provides a queuing infrastructure that provides guaranteed message delivery, security, and priority-based messaging.

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

34.1.3.7 Socket Adapter

The socket adapter enables you to create a client or a server socket, and establish a connection. This adapter enables you to model standard or nonstandard protocols for communication over TCP/IP sockets. The transported data can be text or binary in format.

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

34.1.3.8 Third Party Adapter

The third party adapter enables you to integrate third-party adapters such as PeopleSoft, SAP, and others into a SOA composite application. These third-party adapters produce artifacts (WSDLs and JCA files) that can configure a JCA adapter.

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

34.1.4 Oracle Applications Adapter

The Oracle applications adapter provides connectivity to Oracle Applications. The adapter supports all modules of Oracle Applications in Release 12 and Release 11*i*, including selecting custom integration interface types based on the version of Oracle E-Business Suite.

For more information, see *Oracle Fusion Middleware Adapter for Oracle Applications User's Guide*.

34.1.5 Oracle BAM

The Oracle BAM adapter enables you to integrate Java EE applications with Oracle BAM Server to send data.

Dragging an Oracle BAM adapter into a swimlane of the SOA Composite Editor invokes the Adapter Configuration Wizard for specifying configuration properties.

For more information, see Part X, "Using Oracle Business Activity Monitoring" and *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring*.

34.1.6 Oracle B2B

The Oracle B2B service enables you to browse B2B metadata in the MDS repository and select document definitions.

Oracle B2B is an e-commerce gateway that enables the secure and reliable exchange of transactions between an organization and its external trading partners. Oracle B2B and Oracle SOA Suite are designed for e-commerce business processes that require process orchestration, error mitigation, and data translation and transformation within an infrastructure that addresses the issues of security, compliance, visibility, and management.

Dragging Oracle B2B into a swimlane of the SOA Composite Editor invokes the B2B Configuration Wizard for specifying configuration properties.

For more information, see *Oracle Fusion Middleware User's Guide for Oracle B2B*.

34.1.7 ADF-BC Services

The ADF-BC service enables you to integrate Oracle Application Development Framework (ADF) applications using service data objects (SDOs) with SOA composite applications.

Dragging an ADF-BC Service into a swimlane of the SOA Composite Editor invokes the Create ADF-BC Service dialog for specifying configuration properties.

For more information about ADF, see

- Section 6.2, "Delegating XML Data Operations to Data Provider Services"
- Section 6.3, "Using Standalone SDO-based Variables"
- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*

34.1.8 EJB Services

The EJB service enables Enterprise JavaBeans and SOA composite applications to interact by passing SDO parameters (uses a WSDL file to define the interface) or Java interfaces (does not use a WSDL file to define the interface).

SDOs enable you to modify business data regardless of how it is physically accessed. Knowledge is not required about how to access a particular back-end data source to use SDO in a SOA composite application. Consequently, you can use static or dynamic programming styles and obtain connected and disconnected access.

Enterprise JavaBeans are server-side domain objects that fit into a standard component-based architecture for building enterprise applications with Java. These objects become distributed, transactional, and secure components.

Java interfaces eliminate the need for WSDL file definitions. This type of integration provides support with the following objects:

- Native Java objects
- Java Architecture for XML Binding (JAXB)

Dragging an EJB service into a swimlane of the SOA Composite Editor invokes the Create EJB Service dialog for specifying configuration properties.

For more information, see Chapter 35, "Integrating Enterprise JavaBeans with SOA Composite Applications."

34.1.9 Direct Binding Services

The direct binding service uses the Direct Binding Invocation API to invoke a SOA composite application in the inbound direction and exchange messages over a remote method invocation (RMI). This option supports the propagation of both identities and transactions across JVMs and uses the T3 optimized path. Both synchronous and asynchronous invocation patterns are supported.

You can also invoke an Oracle Service Bus (OSB) flow or another SOA composite application in the outbound direction.

Dragging a direct binding service into a swimlane of the SOA Composite Editor invokes the Create Direct Binding Service dialog for specifying configuration properties.

For more information about direct binding, see Chapter 36, "Using the Direct Binding Invocation API."

For information about the Direct Binding Invocation API, see *Oracle Fusion Middleware Infrastructure Management Java API Reference for Oracle SOA Suite*.

For more information about OSB, see *Oracle Fusion Middleware Developer's Guide for Oracle Service Bus*.

34.2 Introduction to Integrating a Binding Component in a SOA Composite Application

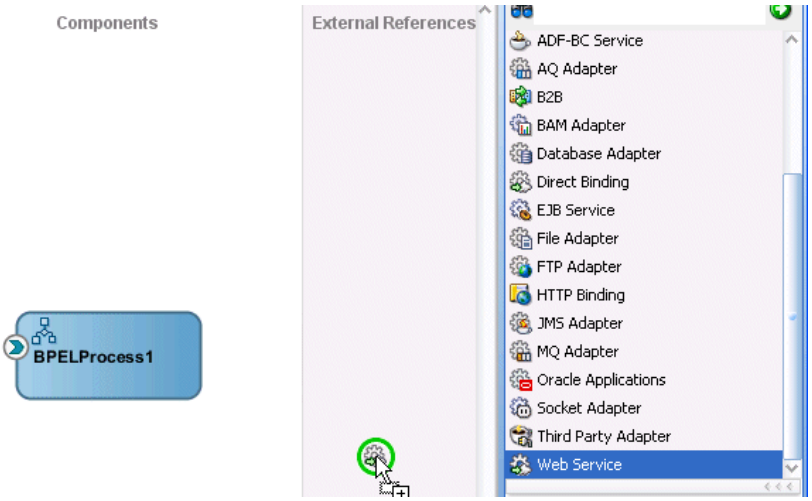
You integrate a binding component with a SOA composite application by dragging it from the Component Palette.

34.2.1 How to Integrate a Binding Component in a SOA Composite Application

1. From the **Service Adapters** section of the Component Palette, drag a binding component to the appropriate swimlane. The swimlane in which to drag the component is based on the action you want to perform.
   - If you want to provide the outside world with an entry point to the SOA composite application, drag the binding component to the **Exposed Services** swimlane.
   - If you want to enable messages to be sent from the SOA composite application to external services in the outside world, drag the binding component to the **External References** swimlane.

Figure 34–6 shows a web service being dragged into the composite. This action invokes a dialog for specifying various configuration properties.

Figure 34–6 Integration of a Web Service Binding Component into a Composite



For more information about adding binding components, see Section 2.3, "Adding Service Binding Components" and Section 2.4, "Adding Reference Binding Components."

34.2.2 How to Use ADF Binding to Invoke a Composite Application from a JSP/Java Class

If a SOA composite application uses web service binding to define an endpoint reference, the composite cannot be invoked from a JSP/Java class. WS binding is defined with the `binding.ws port="" location=""` tag in the `composite.xml` file. Example 34–2 provides details.

Example 34–2 WS Binding Definition

```
<service name="client_ep" ui:wsdlLocation="BPEL.wsdl">
    <interface.wsdl interface="http://xmlns.oracle.com/Application/Project/
      BPEL#wsdl.interface(BPEL)"/>
    <binding.ws port="http://xmlns.oracle.com/App/BPELProj/
      BPELProcess#wsdl.endpoint(bpel_client_ep/BPELProcess_pt)"/>
  </service>
```

Instead, use ADF binding. After deployment of the composite with ADF binding, invocation from a JSP/Java class is successful. Example 34–3 provides details.

Example 34–3 ADF Binding Definition

```
<service name="client_ep" ui:wsdlLocation="BPEL.wsdl">
    <interface.wsdl interface="http://xmlns.oracle.com/Application/Project/
     BPEL#wsdl.interface(BPEL)"/>
    <binding.adf serviceName="bpel_client" registryName=""/>
  </service>
```

35

Integrating Enterprise JavaBeans with SOA Composite Applications

This chapter describes how to integrate Enterprise JavaBeans with SOA composite applications. Integration is achieved through use of service data object (SDO) parameters or Java interfaces.

This chapter includes the following sections:

> **Note:** Support is provided for Enterprise JavaBeans 3.0 and Enterprise JavaBeans 2.0 references (that is, when calling Enterprise JavaBeans 2.0 beans). Support is not provided for Enterprise JavaBeans 2.0 services (that is, when being called with Enterprise JavaBeans 2.0 beans).

35.1 Introduction to Enterprise JavaBeans Binding Integration with SOA Composite Applications

There are two options for integrating Enterprise JavaBeans with SOA composite applications:

- Through use of SDO-based EJBs (uses a WSDL file to define the interface)
- Through use of Java interfaces (does not use a WSDL file to define the interface)

This chapter describes both options.

You can also use the spring service component to integrate Java interfaces with SOA composite applications. For information about using the spring service component, see Chapter 49, "Integrating the Spring Framework in SOA Composite Applications."

35.1.1 Integration Through SDO-Based EJBs

SDOs enable you to modify business data regardless of how it is physically accessed. Knowledge is not required about how to access a particular back-end data source to use SDOs in a SOA composite application. Consequently, you can use static or dynamic programming styles and obtain connected and disconnected access.

Enterprise JavaBeans are server-side domain objects that fit into a standard component-based architecture for building enterprise applications with Java. These objects become distributed, transactional, and secure components.

Many Oracle SOA Suite interfaces are described by WSDL files. Enterprise JavaBeans interfaces are described by Java interfaces. Invocations between the two are made possible in Oracle SOA Suite by an Enterprise JavaBeans Java interface that corresponds to an Oracle SOA Suite WSDL interface.

Through this interface, Oracle SOA Suite provides support for the following:

- Invoking Enterprise JavaBeans with SDO parameters through an Enterprise JavaBeans reference binding component. In this scenario, a SOA composite application passes SDO parameters to an external Enterprise JavaBeans application.
- Invoking an Enterprise JavaBeans service binding component through Enterprise JavaBeans with SDO parameters. In this scenario, an Enterprise JavaBeans application passes SDO parameters into a SOA composite application.

Figure 35–1 provides an overview.

Figure 35–1 SDO and Enterprise JavaBeans Binding Integration

You use the Create EJB Service dialog in Oracle JDeveloper to define this integration, as described in Section 35.3.1, "How to Integrate SDO-based Enterprise JavaBeans with SOA Composite Applications." This option requires the use of a WSDL file. Once complete, the WSDL interaction is defined in the `composite.xml` file through the `interface.wsdl` entry, as shown in Example 35–1.

Example 35–1 WSDL File Definition Through interface.wsdl Entry

```
<service name="PortfolioService">
        <interface.wsdl
 interface="http://bigbank.com/#wsdl.interface(PortfolioService)" />
    <binding.ejb javaInterface="java.class.ejb.com" serviceId="PortfolioService"
                 jarLocation="soaejb.jar"/>
```

35.1.2 Integration Through Java Interfaces

You can also integrate Enterprise JavaBeans with Oracle SOA Suite through Java interfaces, therefore eliminating the need for WSDL file definitions. This type of integration provides support with the following objects:

- Native Java objects
- Java Architecture for XML Binding (JAXB)

Java interfaces differ from SDO interfaces, which are defined in a WSDL file because of the XML-centric nature of service components such as Oracle BPEL Process Manager, Oracle Mediator, and others. No SDO parameters are required when using Java interfaces.

You use the Create EJB Service dialog in Oracle JDeveloper to define this integration, as described in Section 35.3.2, "How to Integrate Java Interface-based Enterprise JavaBeans with SOA Composite Applications." This option does not require the use of a WSDL file. Once complete, the interaction is defined in the `composite.xml` file through the `interface.java` entry, as shown in Example 35–2.

Example 35–2 Java Interface Definition Through interface.java Entry

```
<service name="PortfolioService">
   <interface.java interface="com.bigbank.services.MyService" />
   binding.ejb uri="MyJNDI" ejb-version="EJB3"/>
```

The Java class must be in the project's loader to be available to the user interface. The class must be in `SCA-INF` to be deployed (not all JAR files in the project class path are deployed). This typically means that the class must be in the `SCA-INF/classes` directory or in a JAR in the `SCA-INF/lib` directory. However, it can also be an interface from the system class path.

For information about JAXB, see *Oracle Fusion Middleware Developer's Guide for Oracle TopLink* and Chapter 49, "Integrating the Spring Framework in SOA Composite Applications."

35.2 Designing an SDO-Based Enterprise JavaBeans Application

This section provides a high-level overview of the steps for designing an Enterprise JavaBeans application. For more information, see the following documentation:

- *Oracle Fusion Middleware Programming Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*
- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- Oracle JDeveloper online help table of contents for the following topics:
  - Enterprise JavaBeans
  - SDO for Enterprise JavaBeans/JPA

  Access the help by selecting **Help** > **Table of Contents** in Oracle JDeveloper.

35.2.1 How to Create SDO Objects Using the SDO Compiler

Select one of the following options for creating SDO objects:

- EclipseLink is an open source, object-relational mapping package for Java developers. EclipseLink provides a framework for storing Java objects in a relational database or converting Java objects to XML documents.

  Use EclipseLink to create SDO objects. For instructions on installing, configuring, and using EclipseLink to create SDO objects, visit the following URL:

  `http://wiki.eclipse.org/EclipseLink/Installing_and_Configuring_EclipseLink`

- Oracle JDeveloper enables you to create an SDO service interface for JPA entities. While this feature is more tailored for use with the Oracle Application Development Framework (ADF) service binding in a SOA composite application, you can also use this feature with the Enterprise JavaBeans service binding in SOA composite applications. The SDO service interface feature generates the necessary WSDL and XSD files. If you use this feature, you must perform the following tasks to work with the Enterprise JavaBeans service binding:
  - Browse for and select this WSDL file in the SOA Resource Browser dialog, which is accessible from the **WSDL URL** field of the Create EJB Service dialog (described in Section 35.3, "Creating an Enterprise JavaBeans Service in Oracle JDeveloper").
  - Add the **BC4J Service Runtime** library to the SOA project. To add this library, double-click the project and select **Libraries and Classpath** to add the library in the Project Properties dialog. You are now ready to design the business logic.

  For more information, see the SDO for Enterprise JavaBeans/JPA topic in the Oracle JDeveloper online help (this includes instructions on how create to an SDO service interface).

35.2.2 How to Create a Session Bean and Import the SDO Objects

To create a session bean and import the SDO objects:

1. Create a simple session bean with the Create Session Bean wizard. For details on using this wizard, see the Creating a Session Bean topic in the Oracle JDeveloper online help.
2. Import the SDO objects into your project through the Project Properties dialog.
3. Add logic and necessary import and library files. In particular, you must import the `Commonj.sdo.jar` file. JAR files can be added in the Libraries and Classpath dialog. This dialog is accessible by double-clicking the project and selecting **Libraries and Classpath** in the Project Properties dialog. You are now ready to design the logic.
4. Expose the method to the remote interface.

35.2.3 How to Create a Profile and an EAR File

To create a profile and an EAR file:

1. Create an Enterprise JavaBeans JAR profile in the Project Properties dialog.
2. Create an application level EAR file in the Application Properties dialog.

35.2.4 How to Define the SDO Types with an Enterprise JavaBeans Bean

An Enterprise JavaBeans bean must define the SDO types. Example 35–3 provides details.

Caution: Where to call `define` can be nontrivial. You must force the types to be defined before remote method invocation (RMI) marshalling must occur and in the right helper context. The EclipseLink SDO implementation indexes the helper instance with the application name or class loader.

When you invoke the Enterprise JavaBeans method, an application name is available to the EclipseLink SDO runtime. The EclipseLink SDO looks up the context using the application name as the key. Ensure that the types are defined when the application name is visible. When an Enterprise JavaBeans static block is initialized, the application name is not created. Therefore, putting the `define` in the static block does not work if you are using the default application name-based context. One way to get the application name initialized is to allocate more than two instance beans using the `weblogic-ejb-jar.xml` file.

Example 35–3 Definition of SDO Types

```
InputStreamReader reader = new InputStreamReader(url.openStream());
StreamSource source = new StreamSource(reader);
List<SDOType> list = ((SDOXSDHelper) XSDHelper.INSTANCE).define(source, null);
```

The `weblogic-ejb-jar.xml` file is the descriptor file that must be added in the deployment jar. The `weblogic-ejb-jar.xml` file is automatically created when you create a session bean. This file must be modified by adding the following entries shown in Example 35–4.

Example 35–4 weblogic-ejb-jar.xml File

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<weblogic-ejb-jar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-ejb-jar
 http://www.bea.com/ns/weblogic/weblogic-ejb-jar/1.0/weblogic-ejb-jar.xsd"
                  xmlns="http://www.bea.com/ns/weblogic/weblogic-ejb-jar">

    <weblogic-enterprise-bean>
    <ejb-name>HelloEJB</ejb-name>
    <stateless-session-descriptor>
      <pool>
        <initial-beans-in-free-pool>2</initial-beans-in-free-pool>
      </pool>
    </stateless-session-descriptor>
  </weblogic-enterprise-bean>

    </weblogic-ejb-jar>
```

Figure 35–2 provides a code example of a session bean with SDO logic defined:

Figure 35–2 Session Bean with Defined SDO Logic

```
package sdo.ejb.employee;

import ...;

@Stateless(name = "SessionEJB", mappedName = "sdo-ejb-SessionEJB")
@Remote
@WebService(portName = "SessionEJBBeanServicePort", endpointInterface = "sdo.ejb.employee.SessionEJB")
public class SessionEJBBean implements SessionEJB {
    public SessionEJBBean() {
            try {
                defineSchema("/", "employee.xsd");
                System.out.println("Successfully initialized!");
                } catch (Exception e) {
                    e.printStackTrace();
                }
        }
        public EmployeeResponse getEmployeeInfo(Employee emp){
            System.out.println("Emp SSN -->" +emp.getEmp().getSSN());
            EmployeeDetails empDetails = emp.getEmp();
            EmployeeResponse response = (EmployeeResponse) DataFactory.INSTANCE.create(EmployeeResponse.class);
            empDetails.setEmployeeType("Full Time");
            empDetails.setSSN(emp.getEmp().getSSN());
            response.setResult(empDetails);
            return response;
        }
        private static List defineSchema(String resourceLoc, String resourceName) throws IOException {

              ClassLoader cl = Thread.currentThread().getContextClassLoader();
              URL url = cl.getResource(resourceLoc + resourceName);
              if (url == null)
                 throw new IOException("Can't read " + resourceLoc + resourceName);

              InputStreamReader reader = new InputStreamReader(url.openStream());
              StreamSource source = new StreamSource(reader);
              return ((SDOXSDHelper) XSDHelper.INSTANCE).define(source, null);
          }
```

35.2.5 How to Use Web Service Annotations

To generate the WSDL file, the Enterprise JavaBeans interface must use the following web service annotations. Use of these annotations is described in JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.0. Visit the following URL for details:

http://www.jcp.org/en/jsr/detail?id=224

In addition, only a document/literal WSDL is currently supported by the Enterprise JavaBeans binding layer.

Table 35–1 describes the annotations to use.

Table 35–1 Annotations

Name	Description
`@javax.jws.WebResult;` `@javax.jws.WebParam;`	Customizes the mapping of an individual parameter to a web service message part and XML element. Both annotations are used to map SDO parameters to the correct XML element from the normalized message payload.
`@javax.jws.Oneway;`	Denotes a method as a web service one-way operation that has only an input message and no output message. The Enterprise JavaBeans binding component does not expect any reply in this case.

Table 35–1 (Cont.) Annotations

Name	Description
`@javax.xml.ws.RequestWrapper;`	Tells the Enterprise JavaBeans binding components whether the deserialized object must be unwrapped or whether a wrapper must be created before serialization.
`@javax.xml.ws.ResponseWrapper;`	An Enterprise JavaBeans interface can be generated from an existing WSDL or obtained by some other means. If the WSDL does not exist, it can be generated.
`@javax.xml.ws.WebFault;`	Maps WSDL faults to Java exceptions. This annotation captures the fault element name used when marshalling the JAXB type generated from the global element referenced by the WSDL fault message.
`@oracle.webservices.PortableWebService`	Specifies the `targetNamespace` and `serviceName` used for the WSDL. For example: `@PortableWebService( targetNamespace = "http://hello.demo.oracle/", serviceName = "HelloService")` The `serviceName` is used as the WSDL file name. If it is not specified in the annotations, the SEI class name is used instead.
Add appropriate method parameter annotations	Adds to control how message elements and types are mapped to the WSDL. For example, if your interface is in `doc/lit/bare` style, add the following annotations to the methods. `@WebMethod @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)`
`@SDODatabinding`	Adds to the interface class to use the existing schema instead of a generated one. For example: `@SDODatabinding(schemaLocation = "etc/HelloService.xsd")`

Example 35–5 provides an example of an Enterprise JavaBeans interface with annotations.

Example 35–5 Enterprise JavaBeans Interface with Annotations

```
@Remote
@PortableWebService(targetNamespace = "http://www.example.org/customer-example",
 serviceName = "CustomerSessionEJBService")
@SDODatabinding(schemaLocation = "customer.xsd")
public interface CustomerSessionEJB {
     @WebMethod(operationName="createCustomer")
   @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
   @WebResult(targetNamespace = "http://www.example.org/customer-example",
 partName = "parameters", name = "customer")
   CustomerType createCustomer();
     @WebMethod(operationName="addPhoneNumber")
   @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
   @WebResult(targetNamespace = "http://www.example.org/customer-example",
 partName = "parameters", name = "customer")
   CustomerType addPhoneNumber(@WebParam(targetNamespace =
 "http://www.example.org/customer-example", partName = "parameters", name =
 "phone-number")PhoneNumber phNumber);
}
```

35.2.6 How to Deploy the Enterprise JavaBeans EAR File

To deploy the EAR file from Oracle JDeveloper:

1. Select the Application context menu to the right of the application name.
2. Select **Deploy** and deploy the EAR file to a previously created application server connection.

35.3 Creating an Enterprise JavaBeans Service in Oracle JDeveloper

This section describes how to create an Enterprise JavaBeans reference binding component or Enterprise JavaBeans service binding component in Oracle JDeveloper. The Enterprise JavaBeans service enables the Enterprise JavaBeans application to communicate with Oracle SOA Suite and Oracle SOA Suite to communicate with remote Enterprise JavaBeans.

This section describes how to create the following types of integrations:

- Integration through an SDO interface
- Integration through a Java interface

35.3.1 How to Integrate SDO-based Enterprise JavaBeans with SOA Composite Applications

You can create the following types of SDO-based Enterprise JavaBeans integrations with SOA composite applications:

- Invoke SDO-based Enterprise JavaBeans from a SOA composite application
- Invoke a SOA composite application from Enterprise JavaBeans using SDO parameters

To integrate SDO-based Enterprise JavaBeans with SOA composite applications:

1. In the SOA Composite Editor, drag the **EJB Service** icon into the appropriate swimlane, as described in Table 35–2.

Table 35–2 Swimlane for EJB Service

To Invoke...	Drag the EJB Service to this Swimlane...
SDO-based Enterprise JavaBeans from a SOA composite application	**External References**
A SOA composite application from Enterprise JavaBeans using SDO parameters	**Exposed Services**

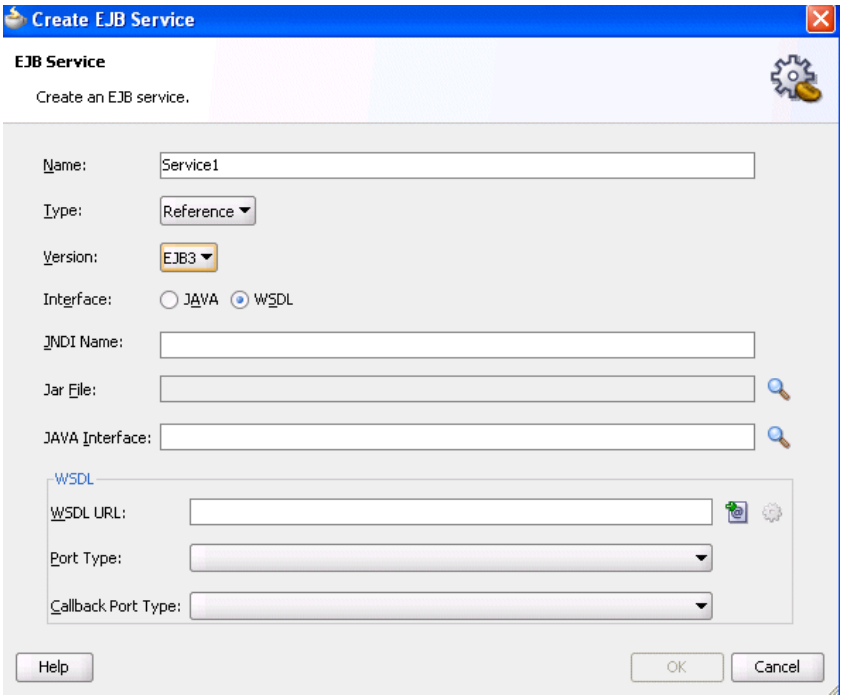
2. In the **Interface** section, click **WSDL**.
3. See the step in Table 35–3 based on the swimlane in which you dragged the **EJB Service**.

Table 35–3 Swimlane Location

If You Dragged the EJB Service to this Swimlane...	Then Go To...
External References	3a
Exposed Services	3b

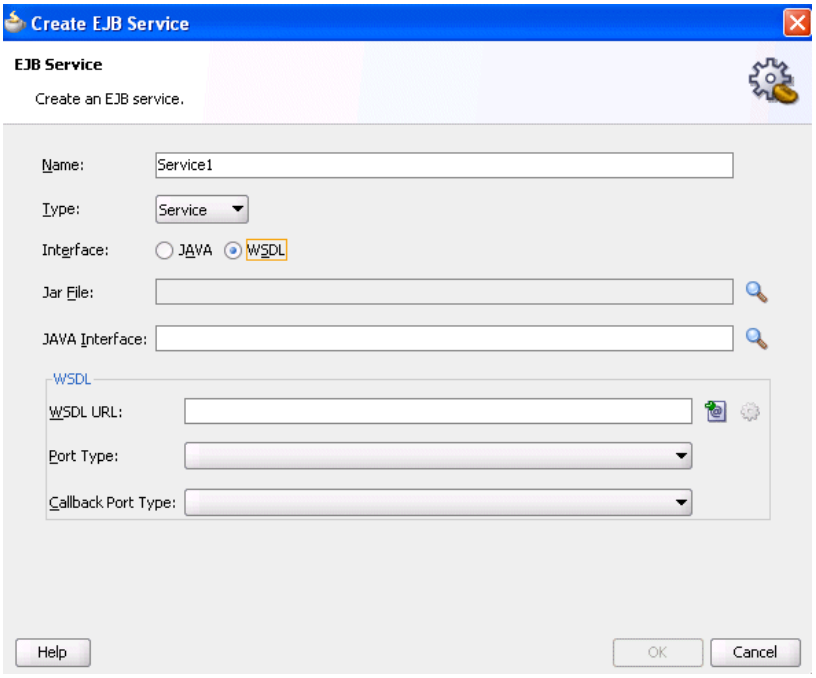
a. View the Create EJB Service dialog that displays in the **External References** swimlane, as shown in Figure 35–3.

Figure 35–3 Create EJB Service in External References Swimlane



b. View the Create EJB Service dialog that displays in the **Exposed Services** swimlane, as shown in Figure 35–4.

Figure 35–4 Create EJB Service in Exposed Services Swimlane



4. Enter values appropriate to your environment. The fields that display differ based on the swimlane in which you dragged the **EJB Service** icon. Table 35–4 provides details.

Table 35–4 Create EJB Service Dialog

Field	Value
Name	Accept the default value or enter a different name.
Type	Displays the following value: ▪ Displays **Reference** if you dragged this icon into the **External References** swimlane. ▪ Displays **Service** if you dragged this icon into the **Exposed Services** swimlane.
Version	Select the version of EJB to support: **EJB2** or **EJB3** (the default selection). **Note**: This field only displays if you dragged the **EJB Service** icon into the **External References** swimlane.
Interface	Select **WSDL**.
JNDI Name	**Note**: This field only displays if you dragged the **EJB Service** icon into the **External References** swimlane. Enter the JNDI name of your Enterprise JavaBeans.
Jar File	Click the **Search** icon to select the EJB JAR file created in Section 35.2, "Designing an SDO-Based Enterprise JavaBeans Application." The SOA Resource Browser dialog searches for and displays JAR files starting in the `SCA-INF/lib` subdirectory of the current project directory. The JAR file includes the interface class and any supporting classes. **Note:** If you select a JAR file outside of the current project, Oracle JDeveloper creates a copy of the JAR file in the **SCA-INF/lib** directory of the current project. When prompted, click **OK** to accept.
Java Interface	Click the **Browse** icon to invoke the Class Browser dialog for selecting the fully qualified Java class name of the previously created Enterprise JavaBeans interface. This class must exist in the selected JAR file. If a JAR file is not specified, it is assumed that the class is in the `/SCA-INF/classes` subdirectory of the current project directory. **Note**: If you use the **Jar File** field, you do *not* need to add a new JAR file to the project by selecting **Project Properties** > **Libraries and Classpath** > **Add JAR/Directory** from the **Application** main menu.
WSDL URL	**Note:** Ensure that you have created the annotations for the Enterprise JavaBeans interface before generating the WSDL file, as described in Section 35.2.5, "How to Use Web Service Annotations." Click the second icon to the right to generate a WSDL file that represents the Enterprise JavaBeans interface. If you created SDO objects through Oracle JDeveloper, as described in Section 35.2.1, "How to Create SDO Objects Using the SDO Compiler," ensure that you select the WSDL file that was automatically generated with this option.
Port Type	Select the port type.
Callback Port Type	Select the callback port type (for asynchronous services).

5. Click **OK**.

35.3.2 How to Integrate Java Interface-based Enterprise JavaBeans with SOA Composite Applications

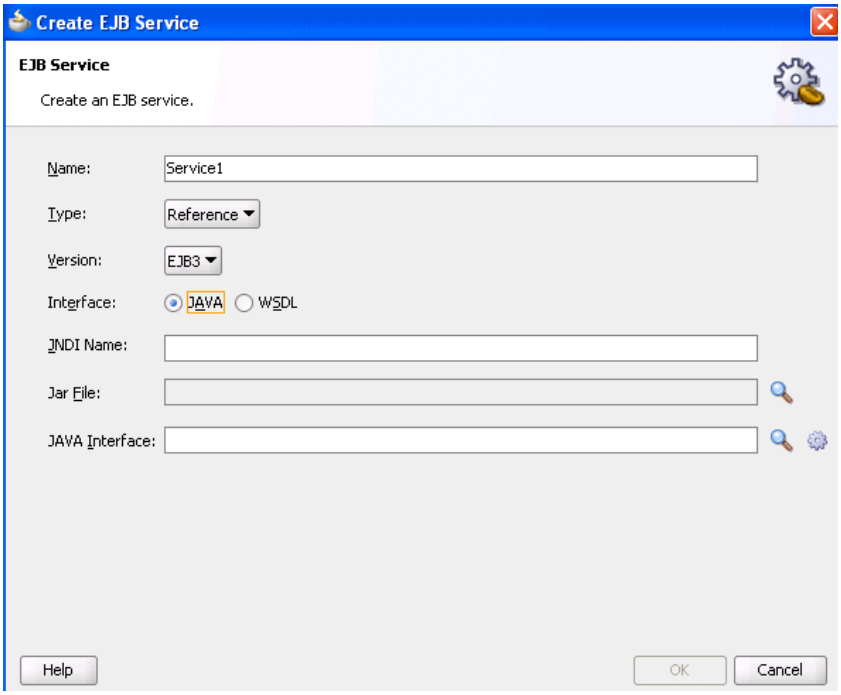
You can create the following types of Java interface-based Enterprise JavaBeans integrations with SOA composite applications:

- Invoke Java interface-based Enterprise JavaBeans from a SOA composite application
- Invoke a SOA composite application from Enterprise JavaBeans using a Java interface

To integrate Java interface-based Enterprise JavaBeans with SOA composite applications:

1. Drag an **EJB Service** icon into the appropriate swimlane:
   - To invoke an Enterprise JavaBeans reference binding component from a SOA composite application, drag the icon to the **External References** swimlane.
   - To invoke a SOA composite application from an Enterprise JavaBeans service binding component, drag the icon to the **Exposed Services** swimlane.
2. In the **Interface** section, click **Java** (if it is not already selected).
3. The Create EJB Service dialog displays the fields shown in Figure 35–5.

Figure 35–5 Create EJB Service for Java Interface



4. Enter the details shown in Table 35–5. The fields are the same regardless of the swimlane in which you dragged the **EJB Service** icon.

Table 35–5 Create EJB Service Dialog

Field	Value
Name	Accept the default value or enter a different name.
Type	Displays the following value: ■ Displays **Reference** if you dragged this icon into the **External References** swimlane. ■ Displays **Service** if you dragged this icon into the **Exposed Services** swimlane.
Version	Select the version of EJB to support: **EJB2** or **EJB3** (the default selection). **Note**: This field only displays if you dragged the **EJB Service** icon into the **External References** swimlane.
Interface	Select **Java**.
JNDI Name	Enter the JNDI name of your Enterprise JavaBeans.
Jar File	Click the **Search** icon to select the EJB JAR file created in Section 35.2, "Designing an SDO-Based Enterprise JavaBeans Application." The SOA Resource Browser dialog searches for and displays JAR files starting in the `SCA-INF/lib` subdirectory of the current project directory. The JAR file includes the interface class and any supporting classes. **Note:** If you select a JAR file outside of the current project, Oracle JDeveloper creates a copy of the JAR file in the **SCA-INF/lib** directory of the current project. When prompted, click **OK** to accept.
Java Interface	Select one of the following options. ■ Enter the Java interface manually. ■ Click the **Browse for Class File** icon to invoke the Class Browser dialog for selecting the Java interface. The class must be available in the runtime classpath. There several ways to make the class available in the runtime classpath. One method is to put the class in the `SCA-INF/classes` directory or in a JAR file in the `SCA-INF/lib` directory at design time to ensure that it gets deployed. However, it can also be an interface from the system class path. There are several ways to make the class available at runtime, but one way is to put the class or JAR into `SCA-INF` at design time so that it gets deployed. **Note**: If you use the **Jar File** field, you do *not* need to add a new JAR file to the project by selecting **Project Properties** > **Libraries and Classpath** > **Add JAR/Directory** from the **Application** main menu. ■ Click the **Generate Java Interface from a WSDL** icon to select the WSDL file from which to generate the Java interface. This option is the same as described in Section 35.3.1, "How to Integrate SDO-based Enterprise JavaBeans with SOA Composite Applications."

5. Click **OK**.

35.4 Designing an SDO-Based Enterprise JavaBeans Client to Invoke Oracle SOA Suite

To invoke an SDO - Enterprise JavaBeans service from Enterprise JavaBeans, you must use the client library. Follow these guidelines to design an Enterprise JavaBeans client.

- Look up the `SOAServiceInvokerBean` from the JNDI tree.
- Get an instance of `SOAServiceFactory` and ask the factory to return a proxy for the Enterprise JavaBeans service interface.
- You can include a client side Enterprise JavaBeans invocation library (`fabric-ejbClient.jar` or the `fabric-runtime.jar` file located in the Oracle JDeveloper home directory or Oracle WebLogic Server) in the Enterprise JavaBeans client application. For example, the `fabric-runtime.jar` file can be located in the *`JDev_Home`*`\jdeveloper\soa\modules\oracle.soa.fabric_11.1.1` directory.

  If the Enterprise JavaBeans application is running in a different JVM than Oracle SOA Suite, the Enterprise JavaBeans application must reference the `ejbClient` library.

Example 35–6 provides an example.

Example 35–6 Enterprise JavaBeans Client Code

```
Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
 "weblogic.jndi.WLInitialContextFactory");
        props.put(Context.PROVIDER_URL, "t3://" + HOSTNAME + ":" + PORT);
        InitialContext ctx = new InitialContext(props);
        SOAServiceInvokerBean invoker =
                (SOAServiceInvokerBean)

ctx.lookup("SOAServiceInvokerBean#oracle.integration.platform.blocks.sdox.ejb.api.
SOAServiceInvokerBean");

        //--  Create a SOAServiceFactory instance
        SOAServiceFactory serviceFactory = SOAServiceFactory.newInstance(invoker);

        //--  Get a dynamice proxy that is essentially a remote reference
        HelloInterface ejbRemote =
 serviceFactory.createService("MyTestEJBService", HelloInterface.class);

        //--  Invoke methods
        Item item = (Item) DataFactory.INSTANCE.create(Item.class);
        item.setNumber(new BigInteger("32"));
        SayHello sayHello = (SayHello)
 DataFactory.INSTANCE.create(SayHello.class);
        sayHello.setItem(item);

        SayHelloResponse response = ejbRemote.sayHello(sayHello);
        Item reply = response.getResult();
```

35.5 Specifying Enterprise JavaBeans Roles

To specify role names required to invoke SOA composite applications from any Java EE application, you add the roles names in the Enterprise JavaBeans service configuration. The Enterprise JavaBeans service checks to see if the caller principal has the security role. Example 35–7 provides details.

Example 35–7 Enterprise JavaBeans Roles

```
<service name="EJBService" ui:wsdlLocation="BPELEJBProcess.wsdl">
    <interface.wsdl
interface="http://xmlns.oracle.com/EJBApplication/EJBProject/BPELEJBProcess#wsdl.i
nt
erface(BPELProcess1)"callbackInterface="http://xmlns.oracle.com/EJBApplication/
EJBProject/BPELEJBProcess#
wsdl.interface(BPELEJBProcessCallback)"/>
<property name="rolesAllowed">Superuser, Admin</property>
    <binding.ejb javaInterface="java.class.ejb.com" serviceId="EJBService"
                 jarLocation="soaejb.jar"/>
</service>
```

35.6 Configuring JNDI Access

This section describes two methods for configuring JNDI access.

35.6.1 How to Create a Foreign JNDI

Follow these guidelines to configure JNDI access.

- You can configure a foreign JNDI provider to link a foreign JNDI tree to your local server instance and access the object as if it is local. See *Oracle Fusion Middleware Programming JNDI for Oracle WebLogic Server*.
- You can also provide JNDI environment variables as the properties for the Enterprise JavaBeans reference, as shown in Example 35–8. An Enterprise JavaBeans binding component enables you to create your own map or use the default `EJBBC` binding component map. Note that the `map` property is optional if you use `EJBBC`. For security reasons, the JNDI security credentials must be stored in a CSF store and referenced as shown in Example 35–8.

Example 35–8 Environment Variables for Enterprise JavaBeans Reference

```
<property name=
"java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</property>
<property name="java.naming.provider.url">t3://host:7001</property>
<property name="oracle.jps.credstore.map">default</property>
<property name="oracle.jps.credstore.key">weblogic</property>
```

The security credential can also be stored in the credential store framework. For more information, see *Oracle Fusion Middleware Security Guide*.

35.6.2 How to Create a Custom CSF Map for JNDI Lookup

If you create your own credential store framework (CSF) map instead of using the default Enterprise JavaBeans BC CSF map, you must modify the *`Domain_Home`*`/config/fmwconfig/system-jazn.data.xml` file and add the permission shown in Example 35–9 to the entry for the `fabric-runtime.jar` permission grant.

Example 35–9 Permission to Add

```
<class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
   <name>>context=SYSTEM,mapName=*,keyName=*</name>
   <actions>*</actions>
</permission>
```

You must then restart Oracle WebLogic Server.

For more information on CSF, see *Oracle Fusion Middleware Security Guide*.

36

Using the Direct Binding Invocation API

This chapter describes the Direct Binding Invocation API and how it can invoke SOA composite applications.

This chapter includes the following sections:

36.1 Introduction to Direct Binding

A common way to invoke a composite is to use SOAP over HTTP. This is enabled by creating a SOAP service for your composite using web service binding. However, you can also use direct binding, which provides a tighter integration alternative. Direct binding enables Java clients to directly invoke composite services, bypassing the intermediate conversion to XML required with web service binding.

Direct binding provides two types of invocation styles:

- Inbound direct binding

  The direct service binding component allows an external client to send messages using the Direct Binding Invocation API, where the Direct Binding Invocation API takes the JNDI connection parameters and creates a connection object on behalf of the client.

- Outbound direct binding (or direct reference binding)

  The direct reference binding component provides support for sending SOA messages directly to external services over RMI. These external services must implement the SOA invocation API (the same as the direct inbound invocation API).

  In the case of direct outbound binding, the connection object is created with the JNDI name of the external service bean configured for the binding.

Direct binding must be associated with the `interface.wsdl`, providing the interface clause and, optionally, the `callbackInterface` clause. The associated WSDL must be imported into the composite.

The service binding component also publishes a modified version of the WSDL that advertises the direct binding.

Direct Service Binding Component

A sample configuration using the direct service binding component is shown in Example 36–1.

Example 36–1 Direct Service Binding Component

```
<service name="direct2">
      <interface.wsdl
interface="http://xmlns.oracle.com/asyncNonConvDocLit#wsdl.interface(asyncNonConvD
ocLit)"
callbackInterface="http://xmlns.oracle.com/asyncNonConvDocLit#wsdl.interface(async
NonConvDocLitCallback)" xmlns:ns="http://xmlns.oracle.com/sca/1.0"/>
      <binding.direct/>
</service>
```

Direct Reference Binding Component

The direct reference binding component requires the following information to connect to a user-provided SOA invoker:

- Properties:

  A set of properties that defines the `DirectConnection` for the end service.

- `ConnectionFactory` class name:

  The `ConnectionFactory` class must implement the `oracle.soa.api.invocation.DirectConnectFactory` interface.

- Address used by the external service:

  This address value is not processed by the binding component, but is passed on to the service bean during invocation.

- `AddressingVersion` (optional):

  The default addressing version used is `2005/08`.

- `useSSLForCallback`:

  Use SSL for the callback JNDI connection. If this flag is set to `true`, then the WSA `replyTo` header instructs the service to call back at an SSL JNDI port.

A sample configuration is shown in Example 36–2.

Example 36–2 Sample Configuration

```
<reference name="HelloReference" ui:wsdlLocation="HelloService.wsdl">
  <interface.wsdl
 interface="http://hello.demo.oracle/#wsdl.interface(HelloInterface)"/>
  <binding.direct connection-factory="oracle.soa.api.JNDIDirectConnectionFactory"
   addressingVersion="http://www.w3.org/2005/08/addressing"
   address="soadirect://syncOut"
   useSSLForCallback="false">
  <property

name="oracle.soa.api.invocation.direct.bean">MyDirectTestServiceBean#directEjb.Tes
tInvoker</property>
  <property

name="java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</property
>
 <property name="java.naming.provider.url">t3://@host:@port</property>
 </binding.direct>
```

```
</reference>
```

The direct binding components support both synchronous and asynchronous invocation patterns. Figure 36–1 describes a sample synchronous invocation pattern and Figure 36–2 describes a sample asynchronous invocation pattern.

Figure 36–1 Sample Synchronous Invocation Patterns

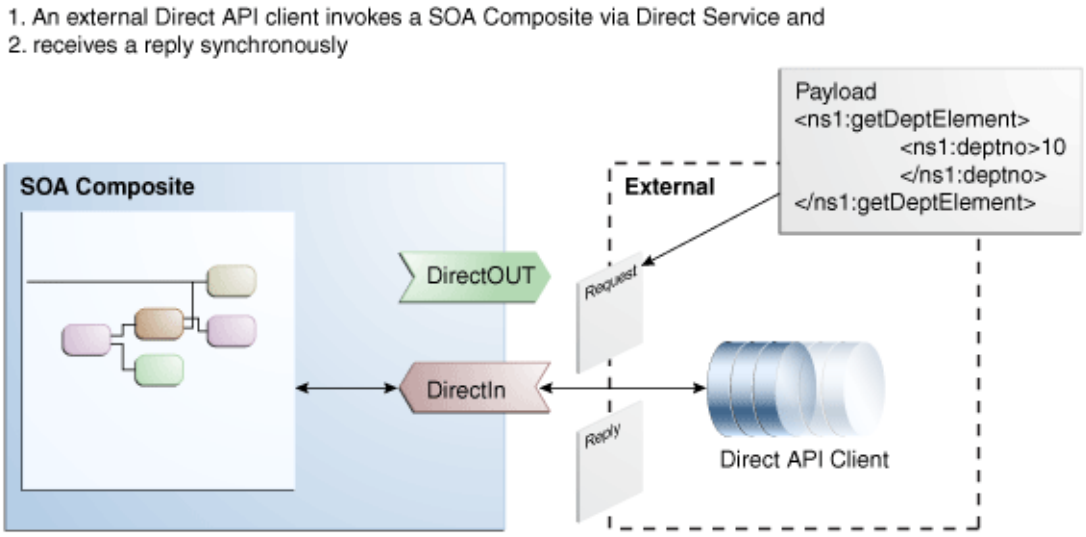
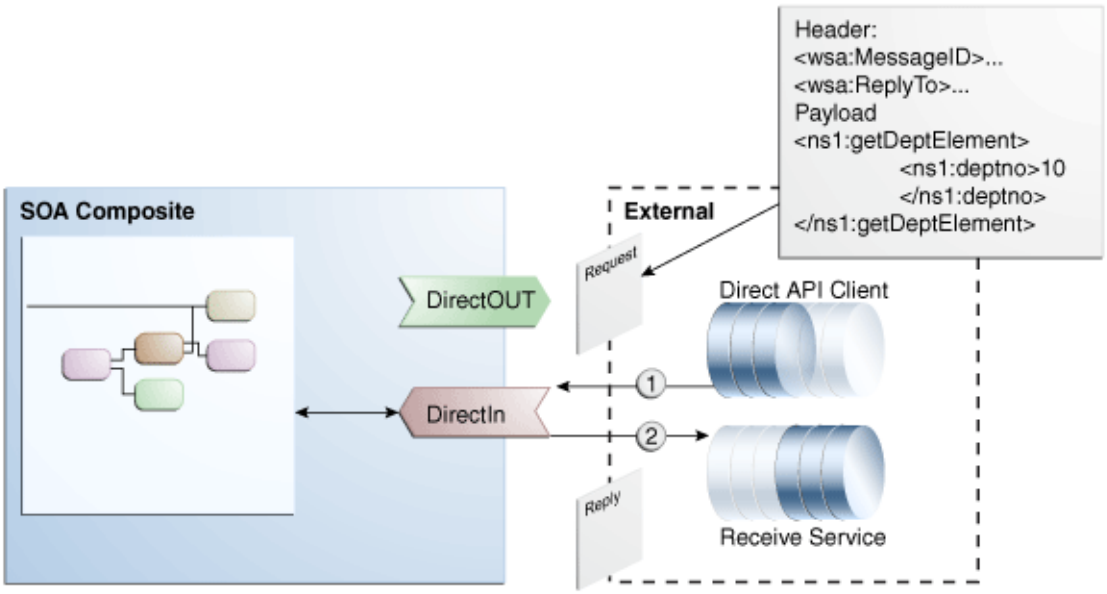


Figure 36–2 Sample Asynchronous Invocation Pattern



36.2 Introduction to the Direct Binding Invocation API

The different packages used in the Direct Binding Invocation API are as follows:

- `oracle.soa.management.facade.Locator`

  The `oracle.soa.management.facade.Locator` interface exposes a method, `createConnection`, which returns a direct connection. The `Locator` exposes the method shown in Example 36–3 for returning the `DirectConnection`.

Example 36–3 oracle.soa.management.facade.Locator

```
import java.util.Map;
public interface DirectConnectionFactory {
    DirectConnection createDirectConnection(CompositeDN compositeDN,
 String serviceName) throws Exception;
```

  You can use the `LocatorFactory` implementation to obtain the `DirectConnection`, as shown in Example 36–4.

Example 36–4 LocatorFactory Implementation

```
Hashtable jndiProps = new Hashtable();
jndiProps.put(Context.PROVIDER_URL, "t3://" + hostname + ':' + portname +
"/soa-infra");
jndiProps.put(Context.INITIAL_CONTEXT_
FACTORY,"weblogic.jndi.WLInitialContextFactory");
jndiProps.put(Context.SECURITY_PRINCIPAL,"weblogic");
jndiProps.put(Context.SECURITY_CREDENTIALS,"welcome1");
jndiProps.put("dedicated.connection","true");
Locator locator = LocatorFactory.createLocator(jndiProps);
CompositeDN compositedn = new CompositeDN(domainName, compositename, version);
String serviceName = "HelloEntry";
return locator.createDirectConnection(compositedn, serviceName);
```

- `oracle.soa.api.invocation.DirectConnection`

  The `DirectConnection` interface invokes a composite service using direct binding. For more information, see *Oracle Fusion Middleware Infrastructure Management Java API Reference for Oracle SOA Suite*.

- `oracle.soa.api.message.Message`

  The `Message` interface encapsulates the data exchanged. For more information, see *Oracle Fusion Middleware Infrastructure Management Java API Reference for Oracle SOA Suite*.

36.2.1 Synchronous Direct Binding Invocation

Direct binding also supports the Synchronous Direct Invocation with the usage of the method shown in Example 36–5.

Example 36–5 Synchronous Direct Invocation

```
<T> Message<T> request(String operationName, Message<T> message)
 throws InvocationException, FaultException
```

36.2.2 Asynchronous Direct Binding Invocation

Asynchronous invocation relies on the WS-Addressing headers set on the message instance. All headers must adhere to the WS-Addressing specification.

The Direct Binding Invocation API allows the clients to specify the WS-Addressing `ReplyTo` SOAP header to communicate a destination by which they can receive responses.

Note: The supported addressing version includes:

- http://www.w3.org/2005/08/addressing
- http://schemas.xmlsoap.org/ws/2004/08/addressing
- http://schemas.xmlsoap.org/ws/2003/03/addressing

An example of the WS-Addressing header used for asynchronous invocation is shown in Example 36–6.

Example 36–6 WS-Addressing Header

```
<wsa:MessageID>D6202742-D9D9-4023-8167-EF0AB81042EC</wsa:MessageID>
   <wsa:ReplyTo xmlns:wsa="http://www.w3.org/2005/08/addressing">
     <wsa:Address>sb://testserver:9001/callback</wsa:Address>
      <wsa:ReferenceParameters>
        <soa:callback xmlns:soa="http://xmlns.oracle.com/soa/direct"
           connection-factory="mytest.MyDirectionConnectionFactory">
        <soa:property name="oracle.soa.api.invocation.direct.bean"
           value="myTest.MyDirectConnectionBean"/>
        <soa:property name="java.naming.provider.url" value="t3://test:8001"/>
        <soa:property name="java.naming.factory.initial"
           value="weblogic.jndi.WLInitialContextFactory"/>
        </soa:callback>
      </wsa:ReferenceParameters>
   </wsa:ReplyTo>
```

Note: You must qualify the callback and its property elements properly with the SOA direct namespace.

The direct binding component is responsible for parsing the addressing headers set on the message instance. In this example, there are two headers: `wsa:MessageID` and `wsa:ReplyTo`. The service binding component makes the following properties available for the internal SOA components:

- `tracking.conversationId` = `D6202742-D9D9-4023-8167-EF0AB81042E`
- `replyToAddress` = `sb://testserver:9001/callback`
- `replyToReferenceParameter:element of WSA:ReferenceParameters`

36.2.3 SOA Direct Address Syntax

The service paths used with the Direct Binding Invocation API follow the SOA direct address pattern:

- `soadirect:/CompositeDN/serviceName`, where `CompositeDN` stands for composite distinguished name

In the SOA direct address, the `CompositeDN` has the following form:

`domainName/compositeName[!compositeVersion[*label]]`

36.2.4 SOA Transaction Propagation

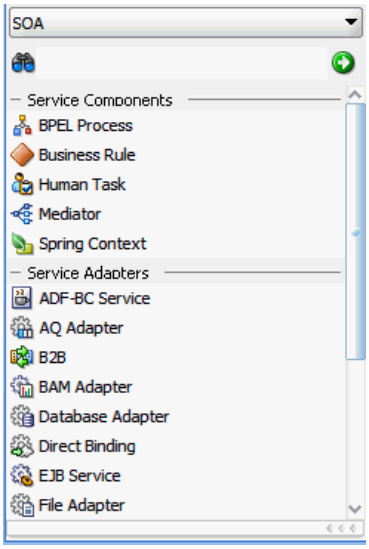
Direct binding supports the SOA transaction propagation feature. You can invoke this feature from the client in the following ways:

- Begin the Java transaction from the client and, after performing all the database operations, perform a commit. You should commit the database operations after a successful commit from the client side.
- Begin the Java transaction from the client side. If a fault is thrown back during any operation in the SOA composite, then roll back the transaction from the client side. This rolls back all the database operations.

36.3 Invoking a SOA Composite Application with the Invocation API

The **Direct Binding** component in Oracle JDeveloper, as shown in Figure 36–3, provides support for exchanging SOA messages with SOA over RMI.

Figure 36–3 Direct Binding Option



Oracle JDeveloper supports creating a direct service binding and a direct reference binding that invokes either an Oracle Service Bus or another SOA composite.

Note: For a client to invoke composite services over direct binding, its class path must include both `soa-infra-mgmt.jar` and `oracle-soa-client-api.jar`.

For more information about the Direct Binding Invocation API, see Section 36.2, "Introduction to the Direct Binding Invocation API."

36.3.1 How to Create an Inbound Direct Binding Service

You can invoke a SOA composite application using the **Direct Binding** option in Oracle JDeveloper.

To create an inbound direct binding service:

1. Open Oracle JDeveloper.
2. From the Component Palette, select **SOA**.
3. From the **Service Adapters** list, drag the **Direct Binding** component into the **Exposed Services** swimlane. The Create Direct Binding dialog appears.

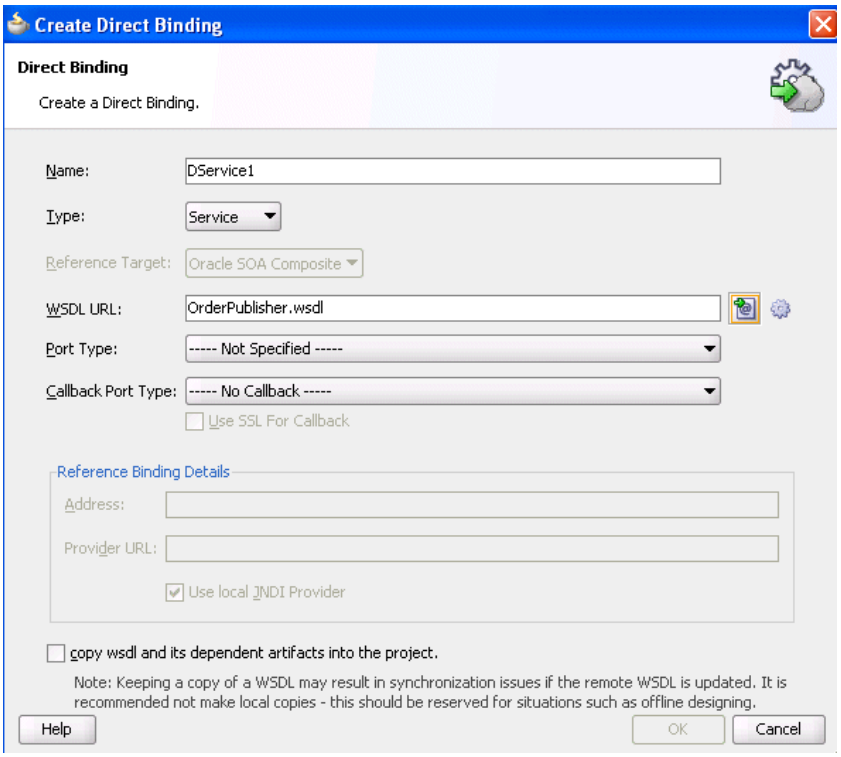
4. Enter the details shown in Table 36–1.

Table 36–1 Direct Binding Service Dialog Fields and Values

Field	Value
Name	Enter a name.
Type	Select **Service** from the list.
Reference Target	This field is disabled when defining this service in the **Exposed Services** swimlane.
WSDL URL	The URL location of the WSDL file. If you have an existing WSDL, then click the **Find Existing WSDLs** option. Otherwise, click **Generate WSDL from schema(s)**.
Port Type	The port type of the WSDL file. You must select a port from the list.
Callback Port Type	The callback port type for asynchronous processes.
Use SSL For Callback	Select to use SSL for the callback.
Address	This field is automatically populated when the WSDL is concrete and it has at least one binding that is direct.
Provider URL	This field is automatically populated when the WSDL is concrete and it has at least one binding that is direct.
Use local JNDI Provider	Select to use the local JNDI provider.
copy wsdl and its dependent artifacts into the project	Deselect this checkbox. If you select this checkbox, the local copies of the WSDL file may result in synchronization issues if a remote WSDL is updated.

When complete, the Create Direct Binding dialog appears as shown in Figure 36–4.

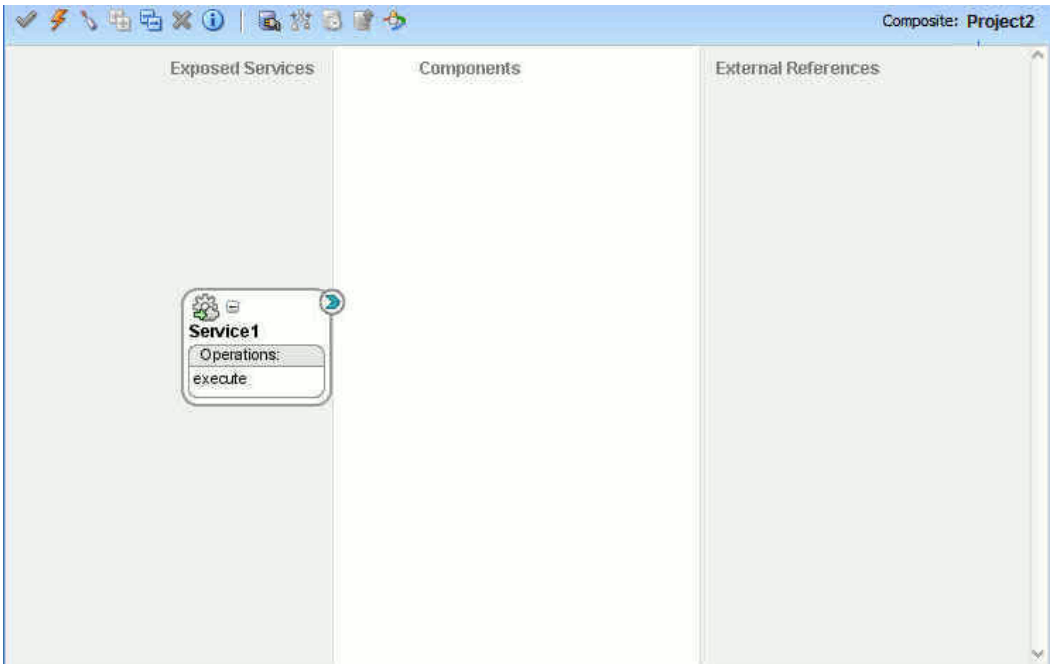
Figure 36–4 Create Direct Binding Dialog



5. Click **OK**.

   The direct binding service displays in the SOA Composite Editor shown in Figure 36–5. The single arrow in a circle indicates that this is a synchronous, one-way, direct binding component.

Figure 36–5 Direct Binding Service



36.3.2 How to Create an Outbound Direct Binding Reference

You can create an outbound direct binding reference, using the **Direct Binding** option in Oracle JDeveloper, to either invoke a composite application or an Oracle Service Bus.

To create an outbound direct binding reference:

1. Open Oracle JDeveloper.
2. From the Component Palette, select **SOA**.
3. From the **Service Adapters** list, drag the **Direct Binding** component into the **External References** swimlane. The Create Direct Binding dialog appears.
4. Enter the details shown in Table 36–2.

Table 36–2 Direct Binding Service Dialog Fields and Values

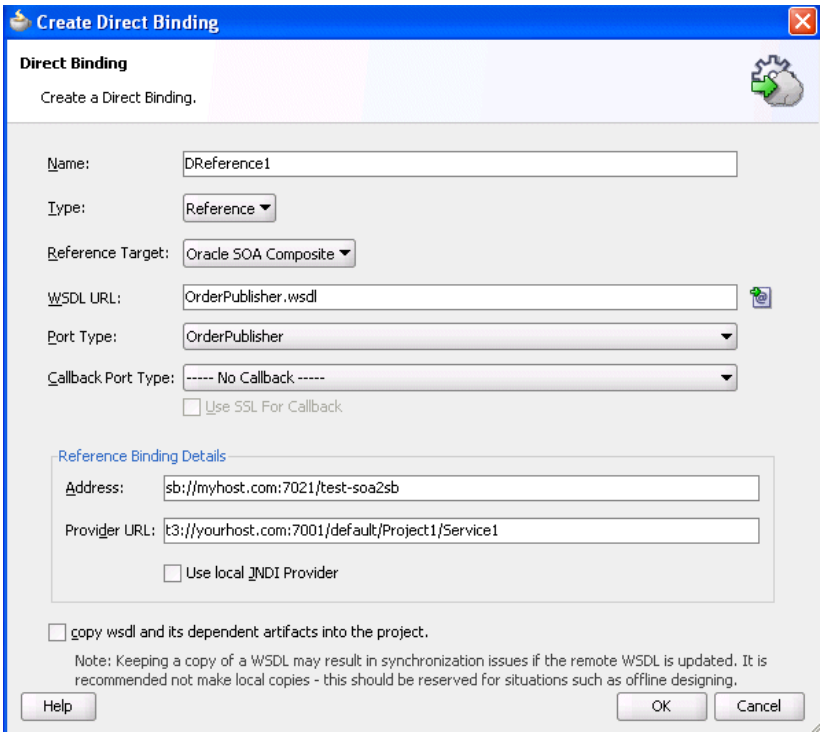
Field	Value
Name	Enter a name.
Type	Select **Reference** from the list.

Table 36–2 (Cont.) Direct Binding Service Dialog Fields and Values

Field	Value
Reference Target	Select the reference target on which you want the direct binding service to operate: ■ **Oracle SOA Composite**: Creates a direct binding with a SOA composite application as a reference target. ■ **Oracle Service Bus**: Creates a direct binding with an Oracle Service Bus as a reference target.
WSDL URL	The URL location of the WSDL file. If you have an existing WSDL, then click the **Find Existing WSDLs** option.
Port Type	The port type of the WSDL file. You must select a port from the list.
Callback Port Type	The callback port type for asynchronous processes.
Use SSL For Callback	Select to use SSL for the callback.
Address	This field is automatically populated when you select a concrete WSDL URL and port type. However, you must manually populate this field if a nonconcrete WSDL is provided.
Provider URL	This field is automatically populated when you select a concrete WSDL URL and port type. However, you must manually populate this field if a nonconcrete WSDL is provided.
Use local JNDI Provider	Select to use the local JNDI provider.
copy wsdl and its dependent artifacts into the project	Deselect this checkbox. If you select this checkbox, the local copies of the WSDL file may result in synchronization issues if a remote WSDL is updated.

When complete, the Create Direct Binding dialog appears as shown in Figure 36–6. For more information about using the Oracle SOA Suite services with Oracle Service Bus, see the Oracle SOA Suite Transport (SOA-DIRECT) chapter in *Oracle Fusion Middleware Developer's Guide for Oracle Service Bus*.

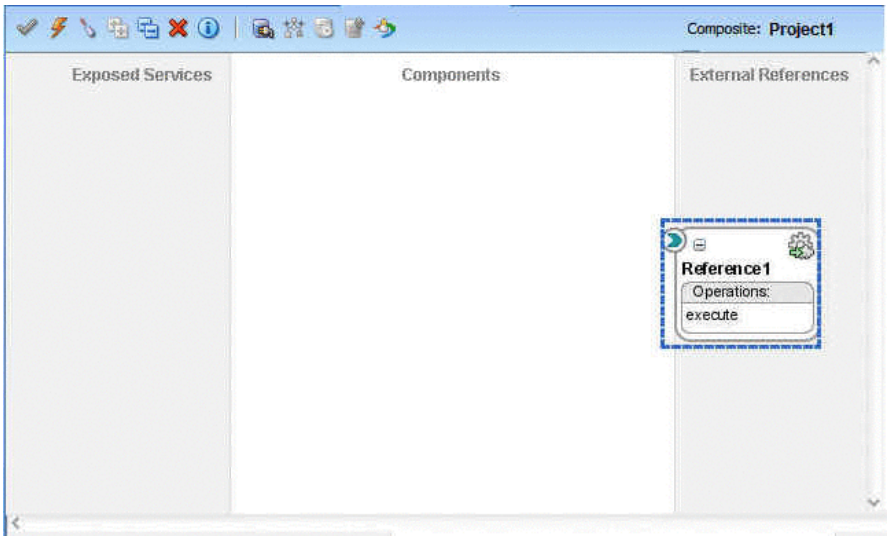
Figure 36–6 Create Direct Binding Dialog



5. Click **OK**.

   The direct binding reference displays in the designer shown in Figure 36–7. The single arrow in a circle indicates that this is a synchronous, one-way direct binding reference component.

Figure 36–7 Direct Binding Reference



36.3.3 How to Set an Identity for J2SE Clients Invoking Direct Binding

J2SE clients can set an identity while invoking direct binding, as shown in Example 36–7.

Example 36–7 Identity Setup for J2SE Clients Invoking Direct Binding

```
public static void main(String[] args) throws Exception {
    String operation = "process";

    // This is the request message XML
    String ns = "http://xmlns.oracle.com/DirectBinding_jws/EchoBPEL/BPELProcess1";
    String payloadXML = "<ns1:process xmlns:ns1=\"" + ns + "\">\n" +
                        "  <ns1:input>wew</ns1:input>\n" +
                        "</ns1:process>";

    String serviceAddress = "soadirect:/default/EchoBPEL!1.0/DService1";

    // Specify the direct binding connection properties
    Map<String, Object> props = new HashMap<String, Object>();
    props.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
    props.put(Context.PROVIDER_URL, "t3://" + hostname + ':' + portname);
    props.put(Context.SECURITY_PRINCIPAL,username);
    props.put(Context.SECURITY_CREDENTIALS, password);

    // Create the direct binding connection, using those context properties
    DirectConnectionFactory factory = JNDIDirectConnectionFactory.newInstance();

    try {
       DirectConnection dc = factory.createConnection(serviceAddress, props);

        // Parse the XML request message
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        Document doc =
          dbf.newDocumentBuilder().parse(new InputSource(new StringReader(payloadXML)));

        // Prepare the payload for inclusion in the Message object
        Map<String, Element> payload = new HashMap<String, Element>();
        payload.put("payload", doc.getDocumentElement());

        Message<Element> request = XMLMessageFactory.getInstance().createMessage(payload);

        Message<Element> response = dc.request(operation, request);
    } finally {
        dc.close();
    }
}
```

36.3.4 What You May Need to Know About Invoking SOA Composites on Hosts with the Same Server and Domain Names

If one SOA composite application invokes another SOA composite application on another host through direct binding, and both composites are on hosts with the same server name and domain name, the invocation fails.

This is because the Oracle WebLogic Server transaction subsystem requires the domain names and server names to be different for transaction management to work properly. The transaction subsystem uses these names to track the location of a server related to a transaction. If the two servers in the invocation have the same name, the transaction subsystem can mistakenly confuse the two servers.

Ensure that you use hosts with separate server names and domain names.

36.4 Samples Using the Direct Binding Invocation API

Example 36–8 through Example 36–10 provide some examples of how the API is used. This section describes how the connection parameter can invoke SOA composite applications over direct binding and how message objects can be modified to invoke a direct binding invocation.

Example 36–8 Usage of a Connection Parameter

```
// The JNDIDirectConnectionFactory can be used to establish SOA instance
// connections for exchanging messages over the direct binding.
DirectConnectionFactory dcFactory = JNDIDirectConnectionFactory.newInstance();

// Connections are created based on the configuration, which is a map of standard
// naming properties, which will be used for the underlying connection lookup.
Map<String, Object> properties = new HashMap<String, Object>();
properties.put(Context.INITIAL_CONTEXT_FACTORY, jndi.WLInitialContextFactory");
properties.put(Context.PROVIDER_URL, "t3://HOST:PORT");
properties.put(Context.SECURITY_PRINCIPAL, USERNAME);
propertie.put(Context.SECURITY_CREDENTIALS, PASSWORD);
DirectConnection conn =
    dcFactory.createConnection("soadirect:/default/MyComposite!1.0/MyService",
                               properties);
```

Example 36–9 Usage of Messages

```
// Messages are created using the MessageFactory
// Message objects are subsequently modified to be used for an invocation.
Message<Element> request = XMLMessageFactory.getInstance().createMessage();

// Define a Map of WSDL part names to matching XML Element objects
Map<String, Element> partData;

Payload<Element> payload = PayloadFactory.createXMLPayload(partData);
request.setPayload(payload);

// One-way invocation
conn.post("onewayoperation", request);

// Request-reply invocation
Message<Element> response = conn.request("requestreplyoperation", request);
```

Example 36–10 Usage of LocatorFactory and DirectConnection Creation

```
Hashtable jndiProps = new Hashtable();
jndiProps.put(Context.PROVIDER_URL, "t3://" + HOST + ':' + PORT);
jndiProps.put(Context.INITIAL_CONTEXT_FACTORY,
              "weblogic.jndi.WLInitialContextFactory");
jndiProps.put(Context.SECURITY_PRINCIPAL,USERNAME);
jndiProps.put(Context.SECURITY_CREDENTIALS, PASSWORD);
Locator locator = LocatorFactory.createLocator(jndiProps);
CompositeDN compositedn = new CompositeDN(domainName, compositename, version);
String serviceName = "HelloEntry";
DirectConnection conn = locator.createDirectConnection(compositedn, serviceName);
```

Part VII

Sharing Functionality Across Service Components

This part describes functionality that can be used by multiple service components.

This part contains the following chapters:

37

Creating Transformations with the XSLT Mapper

This chapter describes how to use the XSLT Mapper. The XSLT Mapper enables you to create data transformations between source schema elements and target schema elements in either Oracle BPEL Process Manager or Oracle Mediator. Version 1.0 of XSLT is supported.

This chapter includes the following sections:

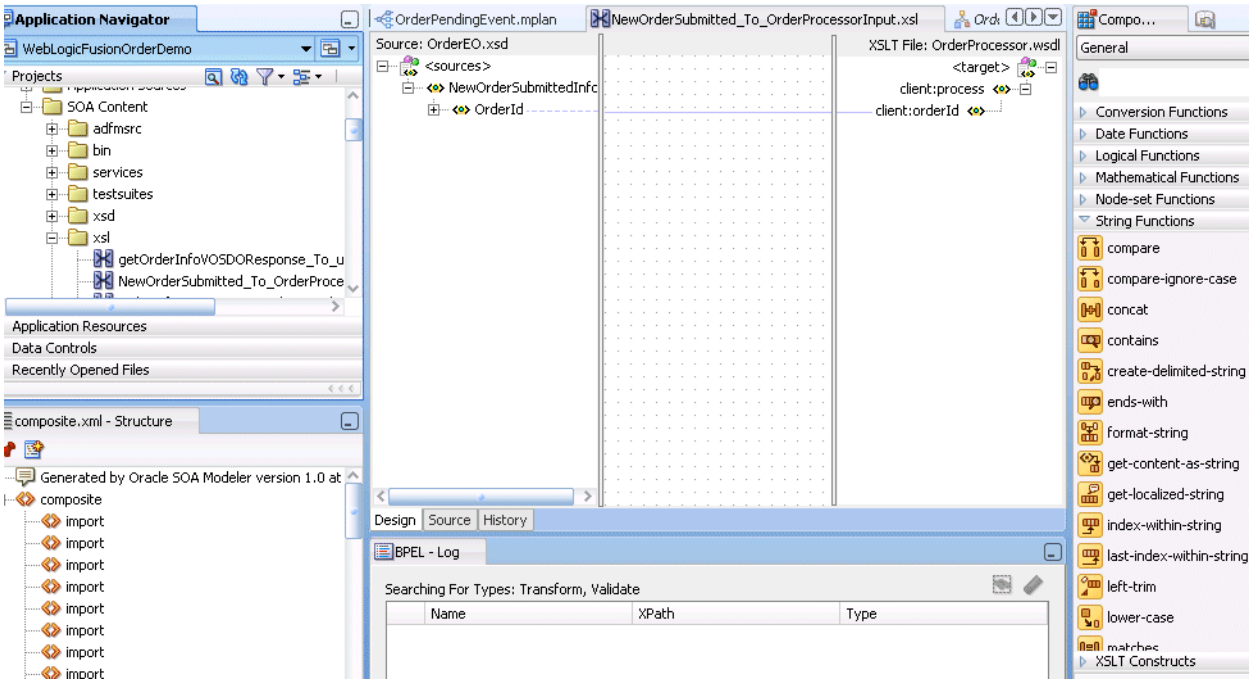
- Section 37.1, "Introduction to the XSLT Mapper"
- Section 37.2, "Creating an XSL Map File"
- Section 37.3, "Designing Transformation Maps with the XSLT Mapper"
- Section 37.4, "Testing the Map"
- Section 37.5, "Demonstrating Features of the XSLT Mapper"

For information on invoking the XSLT Mapper from Oracle BPEL Process Manager, see Section 37.2.1, "How to Create an XSL Map File in Oracle BPEL Process Manager." For information on invoking the XSLT Mapper from Oracle Mediator, see Section 37.2.3, "How to Create an XSL Map File in Oracle Mediator."

37.1 Introduction to the XSLT Mapper

You use the XSLT Mapper to create the contents of a map file. Figure 37–1 shows the layout of the XSLT Mapper.

Figure 37–1 Layout of the XSLT Mapper



The **Source** and the **Target** schemas are represented as trees and the nodes in the trees are represented using a variety of icons. The displayed icon reflects the schema or property of the node. For example:

- An XSD attribute is denoted with an icon that is different from an XSD element.
- An optional element is represented with an icon that is different from a mandatory element.
- A repeating element is represented with an icon that is different from a nonrepeating element, and so on.

The various properties of the element and attribute are displayed in the Property Inspector in the lower right of the XSLT Mapper when the element or attribute is selected (for example, type, cardinality, and so on). The Component Palette in the upper right of Figure 37–1 is the container for all functions provided by the XSLT Mapper. The XSLT Mapper is the actual drawing area for dropping functions and connecting them to source and target nodes.

When an XSLT map is first created, the target tree shows the element and attribute structure of the target XSD. An XSLT map is created by inserting XSLT constructs and XPath expressions into the target tree at appropriate positions. When executed, the XSLT map generates the appropriate elements and attributes in the target XSD.

Editing can be done in design view or source view. When a map is first created, you are in design view. Design view provides a graphical display and enables editing of the map. To see the text representation of the XSLT being created, switch to source view. To switch views, click the **Source** or **Design** tabs at the bottom of the XSLT Mapper.

While in design view, the following pages from the Component Palette can be used:

- **General**: Commonly used XPath functions and XSLT constructs.
- **Advanced**: More advanced XPath functions such as database and cross-reference functions.

- **User Defined**: User-defined functions and templates. This page is visible only when the user has templates in their XSL or user-defined external functions defined through the preferences pages.
- **All Pages**: Provides a view of all functions in one page.
- **My Components**: Contains user favorites and recently-used functions. To add a function to your favorites, right-click the function in the Component Palette and select **Add to Favorites**.

Note: The following functions are only available with Oracle Mediator, and not Oracle BPEL Process Manager, in the XSLT Mapper.

- `getProperty(propertyName as string)`
- `setCompositeInstanceTitle(titleElement)`
- `getComponentInstanceID()`
- `getComponentName()`
- `getCompositeInstanceID()`
- `getCompositeName()`
- `getECID()`

For Oracle BPEL Process Manager, you can use these functions in an assign activity.

While in source view, the **XML** and the **http://www.w3.org/1999/XSL/Transform** pages can be used.

The XSLT Mapper provides three separate context sensitive menus:

- The source panel
- The target panel
- The center panel

Right-click each of the three separate panels to see what the context menus look like.

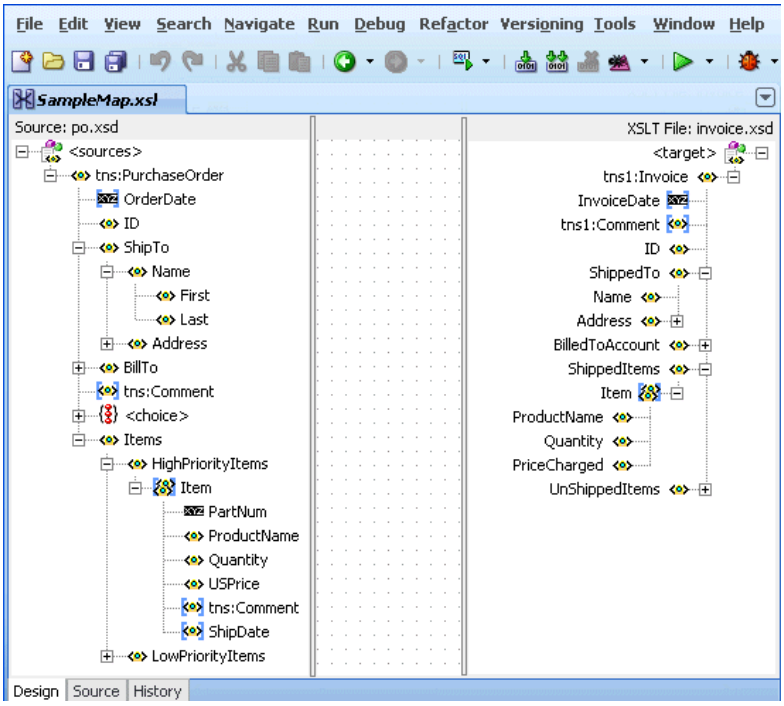
By default, design view shows all defined prefixes for all nodes in the source and target trees. You can elect not to display prefixes by selecting **Hide Prefixes** from the context menu in the center panel of the design view. After prefixes are hidden, select **Show Prefixes** to display them again.

37.1.1 Overview of XSLT Creation

It is important to understand how design view representation of the map relates to the generated XSLT in source view. This section provides a brief example.

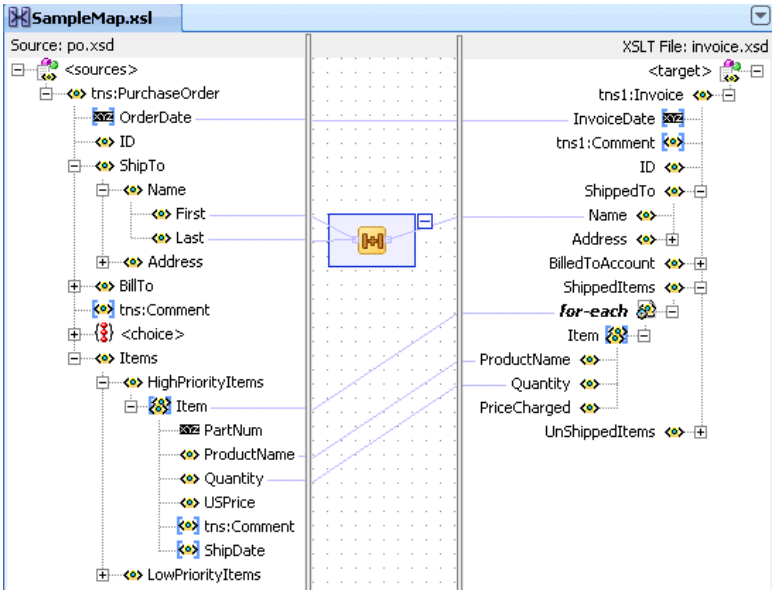
After creating an initial map, the XSLT Mapper displays a graphical representation of the source and target schemas, as shown in Figure 37–2.

Figure 37–2 Source and Target Schemas



At this point, no target fields are mapped. Switching to source view displays an empty XSLT map. XSLT statements are built graphically in design view, and XSLT text is then generated. For example, design view mapping is shown in Figure 37–3.

Figure 37–3 Design View Mapping



The design view results in the generation of the following XSLT statements in source view:

- The **OrderDate** attribute from the source tree is linked with a line to the **InvoiceDate** attribute in the target tree in Figure 37–3. This results in a `value-of` statement in the XSLT, as shown in Example 37–1.

Example 37–1 value-of Statement

```
<xsl:attribute name="InvoiceDate">
  <xsl:value-of select="/ns0:PurchaseOrder/@OrderDate"/>
</xsl:attribute>
```

- The **First** and **Last** name fields from the source tree in Figure 37–3 are concatenated using an XPath **concat** function. The result is linked to the **Name** field in the target tree. This results in the XSLT statement shown in Example 37–2:

Example 37–2 concat Function

```
<Name>
    <xsl:value-of select="concat(/ns0:PurchaseOrder/ShipTo/Name/First,
    /ns0:PurchaseOrder/ShipTo/Name/Last)"/>
</Name>
```

- Note the inserted XSLT **for-each** construct in the target tree in Figure 37–3. For each **HighPriorityItems/Item** element in the source tree, a **ShippedItems/Item** element is created in the target tree and **ProductName** and **Quantity** are copied for each. The XSLT syntax shown in Example 37–3 is generated:

Example 37–3 for-each Construct

```
<xsl:for-each
 select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item">
  <Item>
    <ProductName>
      <xsl:value-of select="ProductName"/>
    </ProductName>
    <Quantity>
      <xsl:value-of select="Quantity"/>
    </Quantity>
   </Item>
</xsl:for-each>
```

The line linking **Item** in the source tree to the **for-each** construct in the target tree in Figure 37–3 determines the XPath expression used in the **for-each** select attribute. In general, XSLT constructs have a select or test attribute that is populated by an XPath statement typically referencing a source tree element.

Note that the XPath expressions in the **value-of** statements beneath the **for-each** construct are relative to the XPath referenced in the **for-each**. In general, the XSLT Mapper creates relative paths within **for-each** statements.

If you must create an absolute path within a **for-each** construct, you must do this within source view. When switching back to design view, it is remembered that the path is absolute and the XSLT Mapper does not modify it.

Note: In Example 37–3, the fields `ProductName` and `Quantity` are required fields in both the source and target. If these fields are optional in the source and target, it is a good practice to insert an `xsl:if` statement around these mappings to test for the existence of the source node. If this is not done, and the source node does not exist in the input document, an empty node is created in the target document. For example, if `ProductName` is optional in both the source and target, then map them as follows:

```
<xsl:if test="ProductName">
    <ProductName>
      <xsl:value-of select="ProductName"/>
    </ProductName>
</xsl:if>
```

The entire XSLT map generated for this example is shown in Example 37–4:

Example 37–4 Entire XSLT Map

```
<xsl:template match="/">
  <tns1:Invoice>
    <xsl:attribute name="InvoiceDate">
      <xsl:value-of select="/ns0:PurchaseOrder/@OrderDate"/>
    </xsl:attribute>
    <ShippedTo>
      <Name>
        <xsl:value-of select="concat
(/ns0:PurchaseOrder/ShipTo/Name/First,/ns0:PurchaseOrder/ShipTo/Name/Last)"/>
      </Name>
    </ShippedTo>
    <ShippedItems>
      <xsl:for-each select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item">
        <Item>
          <ProductName>
            <xsl:value-of select="ProductName"/>
          </ProductName>
          <Quantity>
            <xsl:value-of select="Quantity"/>
          </Quantity>
        </Item>
      </xsl:for-each>
    </ShippedItems>
  </tns1:Invoice>
</xsl:template>
```

Subsequent sections of this chapter describe how to link source and target elements, add XSLT constructs, and create XPath expressions in design view.

37.1.2 Guidelines for Using the XSLT Mapper

- A node in the target tree can be linked only once (that is, you cannot have two links connecting a node in the target tree).
- An incomplete function and expression does not result in an XPath expression in source view. If you switch from design view to source view with one or more incomplete expressions, the Mapper Messages window displays warning messages.

- When you map duplicate elements in the XSLT Mapper, the style sheet becomes invalid and you cannot work in the **Design** view. The Log window shows the error messages when you map an element with a duplicate name. Example 37–5 provides details.

Example 37–5 Duplicate Name Error Messages

```
Error: This Node is Already Mapped  :
"/ns0:rulebase/for-each/ns0:if/ns0:atom/ns0:rel"
  Error: This Node is Already Mapped  :
"/ns0:rulebase/for-each/ns0:if/ns0:atom/choice_1/ns0:ind"
  Error: This Node is Already Mapped  :
"/ns0:rulebase/for-each/ns0:if/ns0:atom/choice_1/ns0:var"
```

Duplicate nodes can be created in design view by surrounding each duplicate node with a **for-each** statement that executes once.

37.2 Creating an XSL Map File

Transformations are performed in an XSL map file in which you map source schema elements to target schema elements. This section describes methods for creating the XSL map file.

Note: You can also create an XSL map file from an XSL style sheet. Click **New** > **SOA Tier** > **Transformations** > **XSL Map From XSL Stylesheet** from the **File** main menu in Oracle JDeveloper.

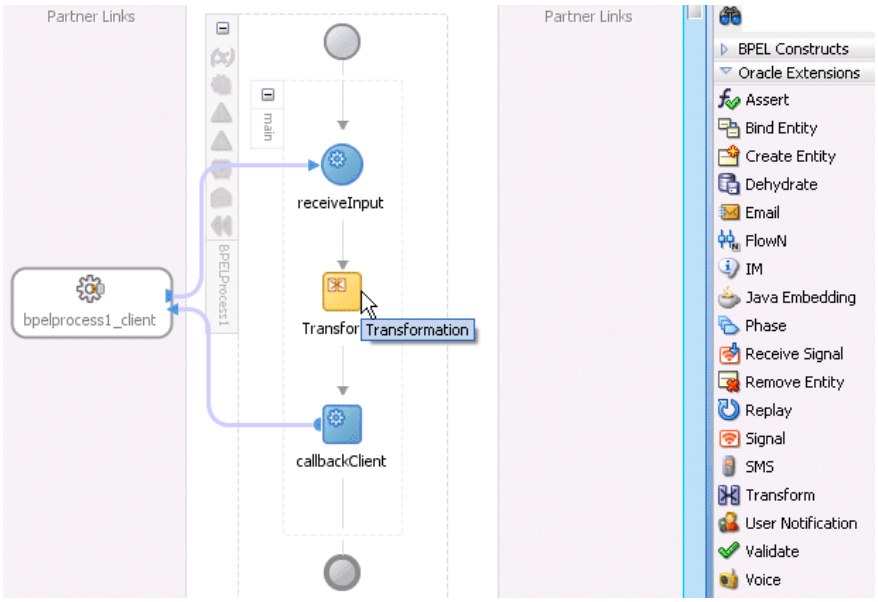
37.2.1 How to Create an XSL Map File in Oracle BPEL Process Manager

A transform activity enables you to create a transformation using the XSLT Mapper in Oracle BPEL Process Manager. This tool enables you to map one or more source elements to target elements. For example, you can map incoming source purchase order schema data to outgoing invoice schema data.

To create an XSL map file in Oracle BPEL Process Manager:

1. From the Component Palette, drag a **transform** activity into your BPEL process diagram. Figure 37–4 provides an example.

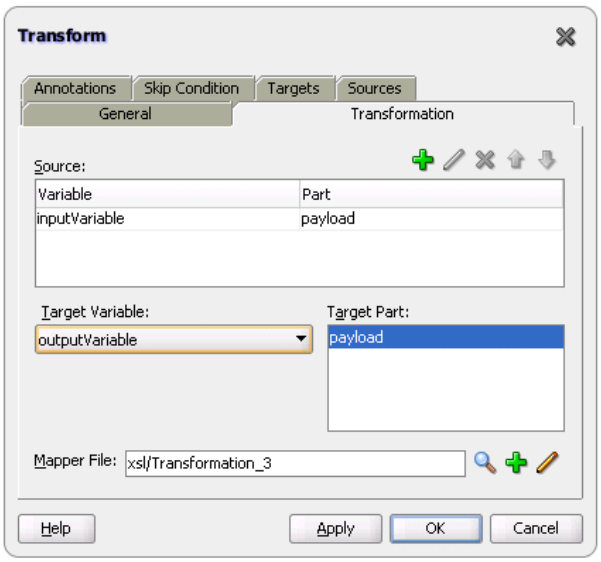
Figure 37–4 Transform Activity



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

   The Transform dialog shown in Figure 37–5 appears.

Figure 37–5 Transform Dialog



3. Specify the following information:

   a. Add source variables from which to map elements by clicking the **Add** icon and selecting the variable and part of the variable as needed (for example, a payload schema consisting of a purchase order request).

> **Note:** You can select multiple input variables. The first variable defined represents the main XML input to the XSL map. Additional variables that are added here are defined in the XSL map as input parameters.

b. Add target variables to which to map elements.

c. Add the target part of the variable (for example, a payload schema consisting of an invoice) to which to map.

4. In the **Mapper File** field, specify a map file name or accept the default name. The map file is the file in which you create your mappings using the XSLT Mapper.

5. Click the **Add** icon (second icon to the right of the **Mapper File** field) to create a mapping. If the file exists, click the **Edit** icon (third icon) to edit the mapping.

   The XSLT Mapper appears.

6. Go to Section 37.1, "Introduction to the XSLT Mapper" for an overview of using the XSLT Mapper.

37.2.2 How to Create an XSL Map File from Imported Source and Target Schema Files in Oracle BPEL Process Manager

> **Note:** If you select a file with a `.xslt` extension such as `xform.xslt`, it opens the XSLT Mapper to create an XSL file named `xform.xslt.xsl`, even though your intension was to use the existing `xform.xslt` file. A `.xsl` extension is appended to *any* file that does not have a `.xsl` extension, and you must create the mappings in the new file. As a work around, ensure that your files first have an extension of `.xsl`. If the XSL file has an extension of `.xslt`, then rename it to `.xsl`.

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

To create an XSL map file from imported source and target schema files in Oracle BPEL Process Manager:

1. In Oracle JDeveloper, select the application project in which you want to create the new XSL map.

2. Import the **po.xsd** and **invoice.xsd** files into the project (for example, in the Structure window of Oracle JDeveloper, right-click **Schemas** and select **Import Schemas**).

3. Right-click the selected project and select **New**.

   The New Gallery dialog appears.

4. In the **Categories** tree, expand **SOA Tier** and select **Transformations**.

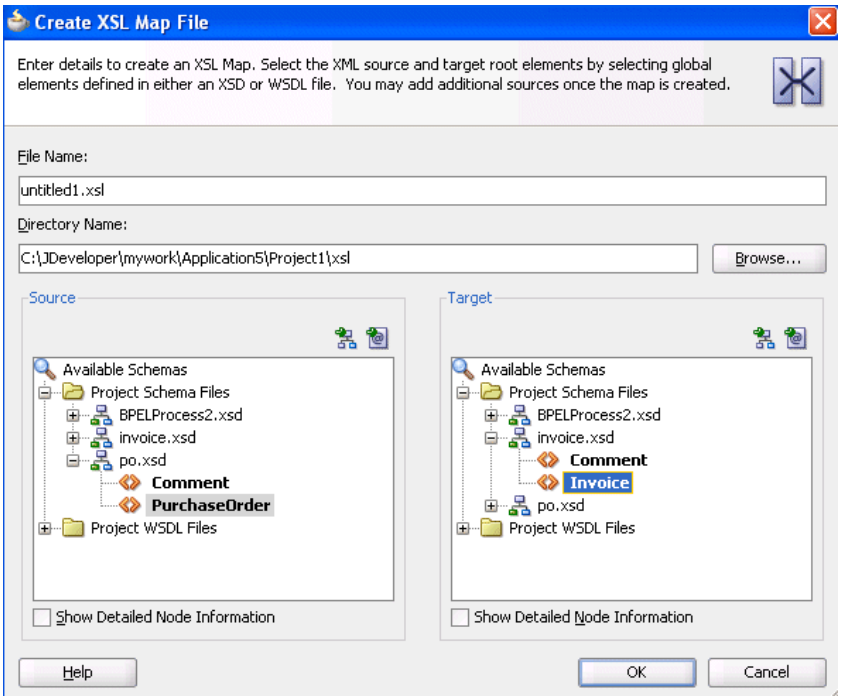
5. In the **Items** list, double-click **XSL Map**.

   The Create XSL Map File dialog appears. This dialog enables you to create an XSL map file that maps a root element of a source schema file or Web Services Description Language (WSDL) file to a root element of a target schema file or WSDL file. Note the following details:

- WSDL files that have been added to the project appear under **Project WSDL Files**.
- Schema files that have been added to the project appear under **Project Schema Files**.
- Schema files that are not part of the project can be imported using the **Import Schema File** facility. Click the **Import Schema File** icon (first icon to the right and above the list of schema files).
- WSDL files that are not part of the project can be imported using the Import WSDL File facility. Click the **Import WSDL File** icon (second icon to the right and above the list of schema files).

6. Enter a name for the XSL map file in the **File Name** field.
7. Select the root element for the source and target trees. In the example in Figure 37–6, the **PurchaseOrder** element is selected for the source root element and the **Invoice** element is selected for the target root element.

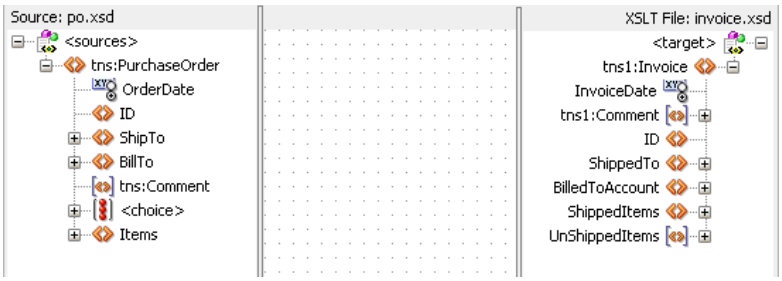
Figure 37–6 Expanded Target Section



8. Click **OK**.

   A new XSL map is created, as shown in Figure 37–7.

Figure 37–7 New XSL Map



9. Save and close the file now or begin to design your transformation. Information on using the XSLT Mapper is provided in Section 37.1, "Introduction to the XSLT Mapper."
10. From the Component Palette, drag a **transform** activity into your BPEL process.
11. Double-click the **transform** activity.
12. Specify the following information:
    a. Add source variables from which to map elements by clicking the **Add** icon and selecting the variable and part of the variable as needed (for example, a payload schema consisting of a purchase order request).

    Note: You can select multiple input variables. The first variable defined represents the main XML input to the XSL map. Additional variables that are added here are defined in the XSL map as input parameters.

    b. Add target variables to which to map elements.
    c. Add the target part of the variable (for example, a payload schema consisting of an invoice) to which to map.
13. To the right of the **Mapper File** field, click the **Search** icon (first icon) to browse for the map file name you specified in Step 6.
14. Click **Open**.
15. Click **OK**.

    The XSLT Mapper displays your XSL map file.
16. Go to Section 37.1, "Introduction to the XSLT Mapper" for an overview of using the XSLT Mapper.

37.2.3 How to Create an XSL Map File in Oracle Mediator

The XSLT Mapper enables you to create an XSL file to transform data from one XML schema to another in Oracle Mediator. After you define an XSL file, you can reuse it in multiple routing rule specifications. This section provides an overview of creating a transformation map XSL file with the XSLT Mapper.

The XSLT Mapper is available from the Application Navigator in Oracle JDeveloper by clicking an XSL file or from the Mediator Editor by clicking the **transformation** icon, as described in the following steps. You can either create a new transformation map or update an existing one.

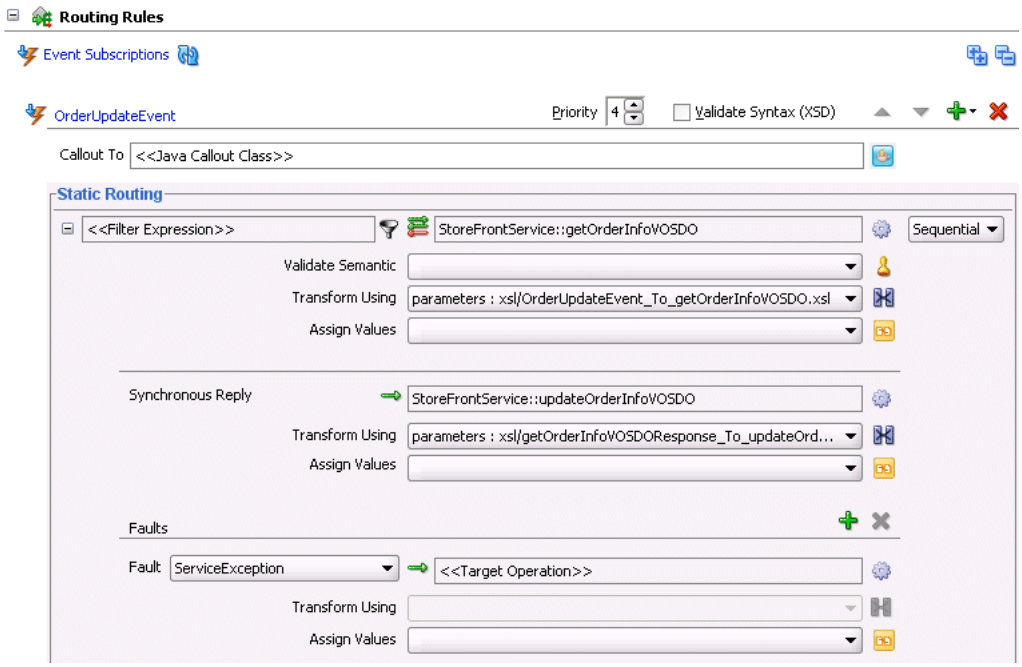
To launch the XSLT Mapper from the Mediator Editor and create or update a data transformation XSL file, follow these steps.

To create an XSL map file in the Mediator Editor:

1. Open the Mediator Editor.
2. To the left of **Routing Rules**, click the **+** icon to open the **Routing Rules** panel.

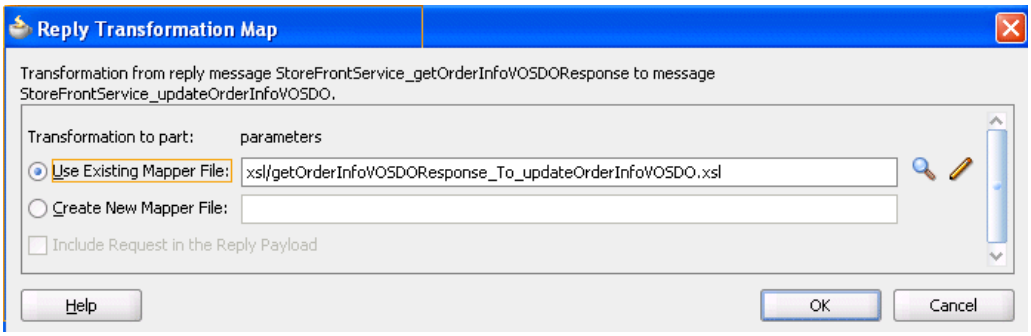
   The **transformation map** icon is visible in the routing rules panel.
3. To the right of the **Transform Using** field shown in Figure 37–8, click the appropriate **transformation map** icon to open the Transformation Map dialog.

Figure 37–8 Routing Rules



The appropriate Transformation Map dialog displays with options for selecting an existing transformation map (XSL) file or creating a new map file. For example, if you select the **transformation map** icon in the **Synchronous Reply** section, the dialog shown in Figure 37–9 appears.

Figure 37–9 Reply Transformation Map Dialog



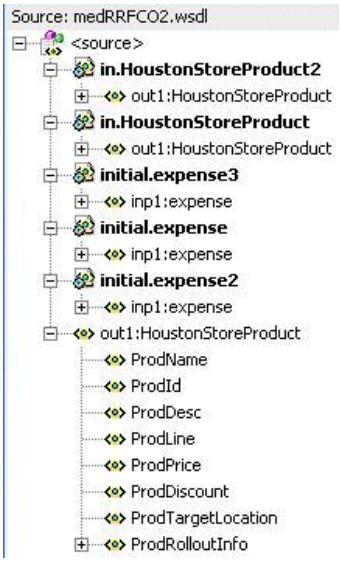
If the routing rule includes a synchronous reply or fault, the Reply Transformation Map dialog or Fault Transformation Map dialog contains the **Include Request in**

the Reply Payload option. When you enable this option, you can obtain information from the request message. The request message and the reply and fault message can consist of multiple parts, meaning you can have multiple source schemas. Callback and callback timeout transformations can also consist of multiple parts.

Each message part includes a variable. For a reply transformation, the reply message includes a schema for the main part (the first part encountered) and an **in.partname** variable for each subsequent part. The include request message includes an **initial.partname** variable for each part.

For example, assume the main reply part is the **out1.HoustonStoreProduct schema** and the reply also includes two other parts that are handled as variables, **in.HoustonStoreProduct** and **in.HoustonStoreProduct2**. The request message includes three parts that are handled as the variables **initial.expense**, **initial.expense2**, and **initial.expense3**. Figure 37–10 provides an example.

Figure 37–10 Reply Part

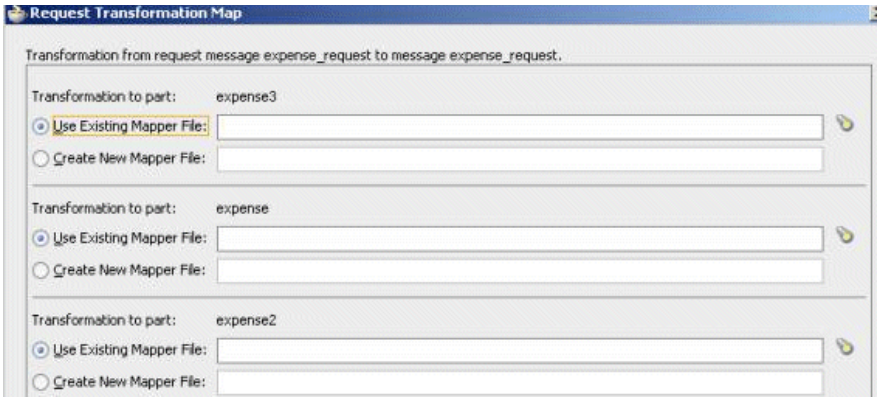


4. Choose one of the following options:

   - **Use Existing Mapper File**, and then click the **Search** icon to browse for an existing XSLT Mapper file (or accept the default value).
   - **Create New Mapper File**, and then enter a name for the file (or accept the default value). If the source message in the WSDL file has multiple parts, variables are used for each part, as mentioned in Step 3. When the target of a transformation has multiple parts, multiple transformation files map to these targets. In this case, the mediator's transformation dialog has a separate panel for each target part. For example, here is a request in which the target has three parts:

   Figure 37–11 provides an example.

Figure 37–11 Request Transformation Map Dialog



5. Click **OK.**

   If you chose **Create New Mapper File**, the XSLT Mapper opens to enable you to correlate source schema elements to target schema elements.

6. Go to Section 37.1, "Introduction to the XSLT Mapper" for an overview of using the XSLT Mapper.

37.2.4 What You May Need to Know About Creating an XSL Map File

XSL file errors do not display during a transformation at runtime if you manually remove all existing mapping entries from an XSL file except for the basic format data. Ensure that you always specify mapping entries. For example, assume you perform the following actions:

1. Create a transformation mapping of input data to output data in the XSLT Mapper.
2. Design the application to write the output data to a file using the file adapter.
3. Manually modify the XSL file and remove all mapping entries except the basic format data. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.fu
nctions.Xpath20"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:ns0="http://xmlns.oracle.com/pcbpel/adapter/file/MediaterDemo/Validation
UsingSchematron/WriteAccounInfoToFile/"
xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.fu
nctions.ExtFunc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dvm="http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue
"
xmlns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:mhdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.mediator.servi
ce.common.functions.GetRequestHeaderExtnFunction"
xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
xmlns:imp1="http://www.mycompany.com/MyExample/NewAccount"
xmlns:tns="http://oracle.com/sca/soapservice/MediaterDemo/ValidationUsingSchem
atron/CreateNewCustomerService"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRe
fXPathFunctions"
xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:inp1="http://www.mycompany.com/MyExample/NewCustomer"
exclude-result-prefixes="xsi xsl tns xsd inp1 ns0 imp1 plt xp20 bpws orcl dvm
hwf mhdr ids xref ora">
</xsl:stylesheet>
```

While the file can still be compiled, the XSL mapping is now invalid.

4. Deploy and create an instance of the SOA composite application.

   During instance creation, an exception error occurs when the write operation fails because it did not receive any input. However, note that no errors displayed during XSL transformation.

37.2.5 What You May Need to Know About Importing a Composite with an XSL File

If you import a SOA archive exported from Oracle Enterprise Manager Fusion Middleware Control into Oracle JDeveloper by selecting **File** > **Import** > **SOA Archive Into SOA Project**, you cannot open any XSL map files because the map headers have been removed.

As a work around, perform the following steps:

1. Select **File** > **New** > **SOA Tier** > **Transformations** > **XSL Map From XSL Stylesheet**, and click **OK**.

   The Create XSL Map File From XSL Stylesheet appears.

2. In the **File Name** field, enter a new map name (for this example, `Transformation_new.xsl`).

3. In the **XSL Stylesheet to Create From** field, enter the name of the map file missing the map headers (for this example, `Transformation_old.xsl`).

4. For the source and target, enter the correct map source and target schema information to use for recovering the map header.

5. After successful creation of the new map file, delete the old map file (`Transformation_old.xsl`).

6. Rename the new map file with the recovered map header to the old map file name to prevent reference breakage (for this example, rename `Transformation_new.xslt` to `Transformation_old.xsl`).

37.2.6 What Happens at Runtime If You Pass a Payload Through Oracle Mediator Without Creating an XSL Map File

If you design a SOA composite application to pass a payload through Oracle Mediator without defining any transformation mapping, Oracle Mediator passes the payload through. However, for the payload to be passed through successfully, you must ensure that your source and target message part names are the same and of the same type. Otherwise, the target reference may fail to execute with error messages such as `Input source like Null` or `Part not found`.

37.2.7 What Happens If You Receive an Empty Namespace Tag in an Output Message

The XML representation from an XSL file may differ from that used in a scenario in which a message is passed through with a transformation being performed or in which an assign activity is used, even though the XMLs are syntactically and semantically the same. For example, if you use a mediator service component to map

an inbound payload that includes an element without a namespace to an outbound payload, you may receive an empty namespace tag in the output message.

```
<Country xmlns="">US</Country>
```

This is the correct behavior. A blank namespace, `xmlns=""`, is automatically added.

37.3 Designing Transformation Maps with the XSLT Mapper

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

37.3.1 How to Add Additional Sources

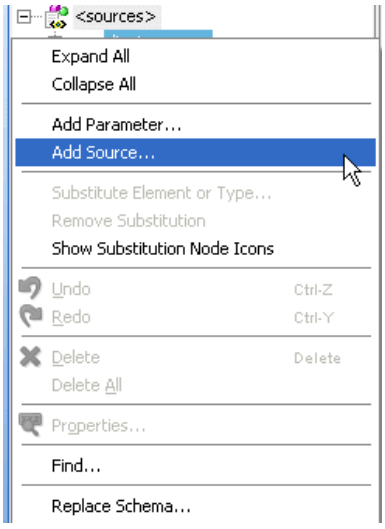
You can add additional sources to an existing XSLT map. These sources are defined as global parameters and have schema files defining their structure. Multiple source documents may be required in certain instances depending upon the logic of the map. For instance, to produce an invoice, the map may need access to both a purchase order and a customer data document as input.

Note that XSL has no knowledge of BPEL variables. When you add multiple sources in XSL design time, ensure that you also add these multiple sources in the transform activity of a BPEL process.

To add additional sources:

1. Right-click the source panel to display the context menu. Figure 37–12 provides details.

Figure 37–12 Context Menu

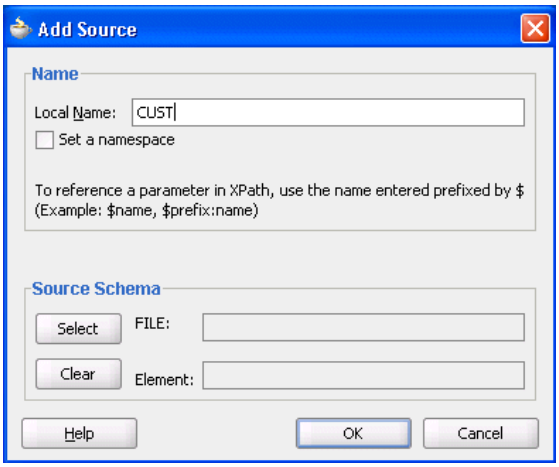


2. Select **Add Source**.

   The Add Source dialog shown in Figure 37–13 appears.

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

Figure 37–13 Add Source Dialog



4. In the **Source Schema** section, click **Select** to select a schema for the new source.

   The Type Chooser dialog appears.

5. Select or import the appropriate schema or WSDL file for the parameter in the same manner as when creating a new XSLT map. For this example, the **Customer** element from the sample **customer.xsd** file is selected.

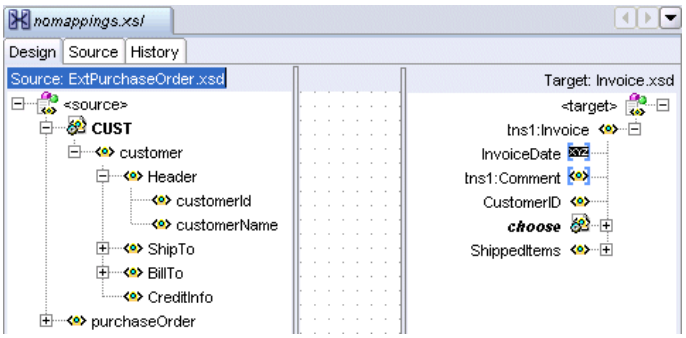
6. Click **OK**.

   The schema definition appears in the **Source Schema** section of the Create Source as Parameter dialog.

7. Click **OK**.

   The selected schema is imported and the parameter appears in the source panel above the main source. The parameter can be expanded as shown in Figure 37–14 to view the structure of the underlying schema.

Figure 37–14 Expanded Parameter



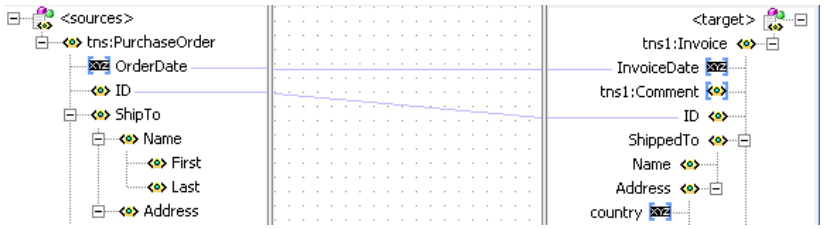
The parameter can be referenced in XPath expressions by prefacing it with a **$**. For example, a parameter named **CUST** appears as **$CUST** in an XPath expression. Nodes under the parameter can also be referenced (for example, **$CUST/customer/Header/customerid**).

37.3.2 How to Perform a Simple Copy by Linking Nodes

To copy an attribute or leaf-element in the source to an attribute or leaf-element in the target, drag the source to the target. For example, copy the element **PurchaseOrder/ID**

to **Invoice/ID** and the attribute **PurchaseOrder/OrderDate** to **Invoice/InvoiceDate**, as shown in Figure 37–15.

Figure 37–15 Linking Nodes



37.3.3 How to Set Constant Values

Perform the following steps to set a constant value.

To set constant values:

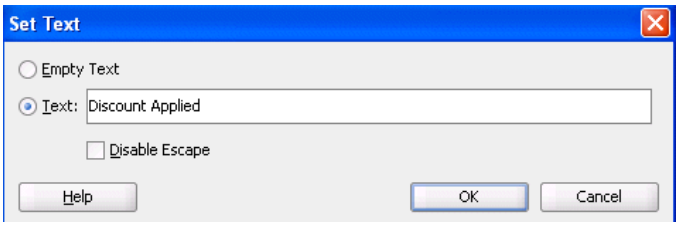
1. Select a node in the target tree.
2. Invoke the context menu by right-clicking the mouse.
3. Select the **Set Text** menu option.

   A menu provides the following selections:

   - **<Empty>**: Enables you to create an empty node.
   - **Enter Text**: Enables you to enter text.
4. Select **Enter Text**.

   The Set Text dialog appears.
5. In the Set Text dialog, enter text (for example, **Discount Applied**, as shown in Figure 37–16).

Figure 37–16 Set Text Dialog



6. Click **OK** to save the text.

   A **T** icon is displayed next to the node that has text associated with it. The beginning of the text that is entered is shown next to the node name.
7. To modify the text associated with the node, right-click the node and select **Edit Text** to invoke the Set Text dialog again.
8. Edit the contents and click **OK**.

   For more information about the fields, see the online Help for the Set Text dialog.
9. To remove the text associated with the node, right-click the node and select **Remove Text**.

37.3.4 How to Add Functions

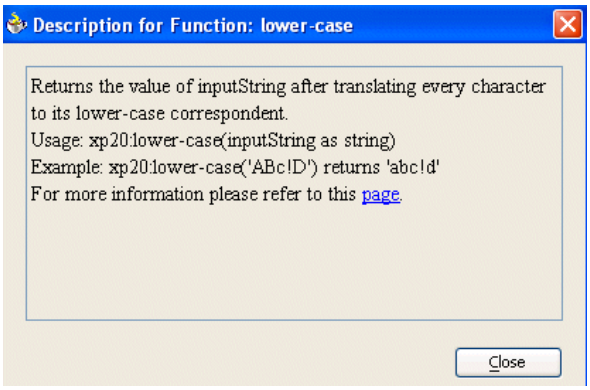
In addition to the standard XPath 1.0 functions, the XSLT Mapper provides many prebuilt extension functions and can support user-defined functions and named templates. The extension functions are prefixed with **oraext** or **orcl** and mimic XPath 2.0 functions.

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

To add functions:

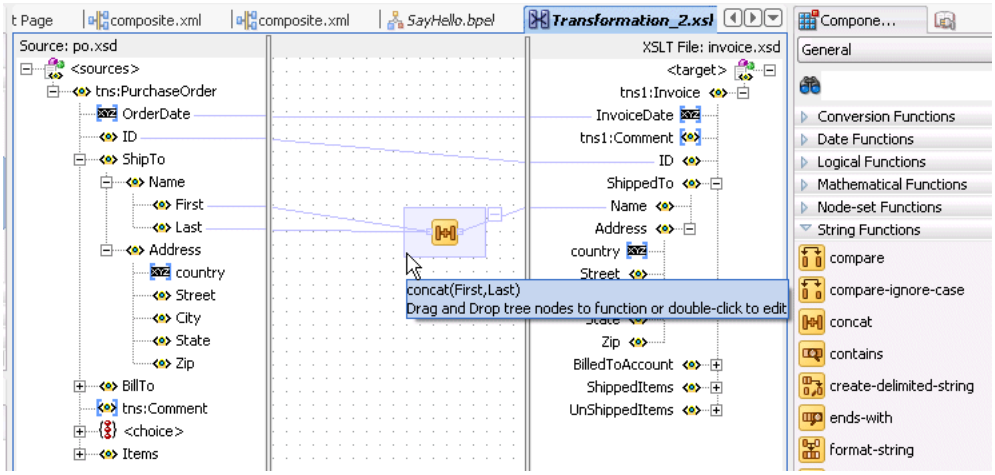
1. From the Component Palette, select a category of functions (for example, **String Functions**).
2. Right-click an individual function (for example, **lower-case**).
3. Select **Help**. A dialog with a description of the function appears, as shown in Figure 37–17. You can also click a link at the bottom to access this function's description at the World Wide Web Consortium at `www.w3.org`.

Figure 37–17 Description of Function



4. Drag a function from the Component Palette to the center panel of the XSLT Mapper. You can then connect the source parameters from the source tree to the function and the output of the function to a node in the target tree. For the following example, drag the **concat** function from the **String** section of the Component Palette to the center panel.
5. Concatenate **PurchaseOrder/ShipTo/Name/First** and **PurchaseOrder/ShipTo/Name/Last**. Place the result in **Invoice/ShippedTo/Name** by dragging threads from the first and last names and dropping them on the input (left) side on the **concat** function.
6. Drag a thread from the **ShippedTo** name and connect it to the output (right) side on the **concat** function, as shown in Figure 37–18.

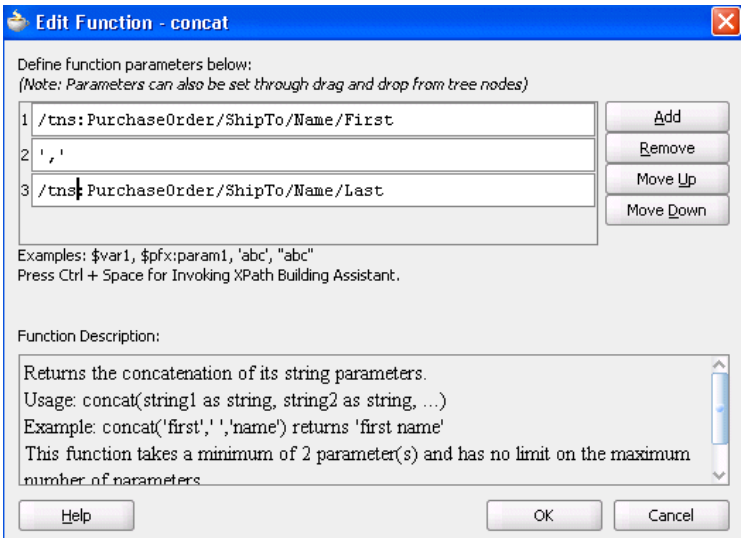
Figure 37–18 Using the Concat Function



37.3.4.1 Editing Function Parameters

To edit the parameters of any function, double-click the function icon to launch the Edit Function dialog. For example, to add a new comma parameter so that the output of the **concat** function used in the previous example is **Last, First**, then click **Add** to add a comma and reorder the parameters to get this output. Figure 37–19 provides details.

Figure 37–19 Editing Function Parameters



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

37.3.4.2 Chaining Functions

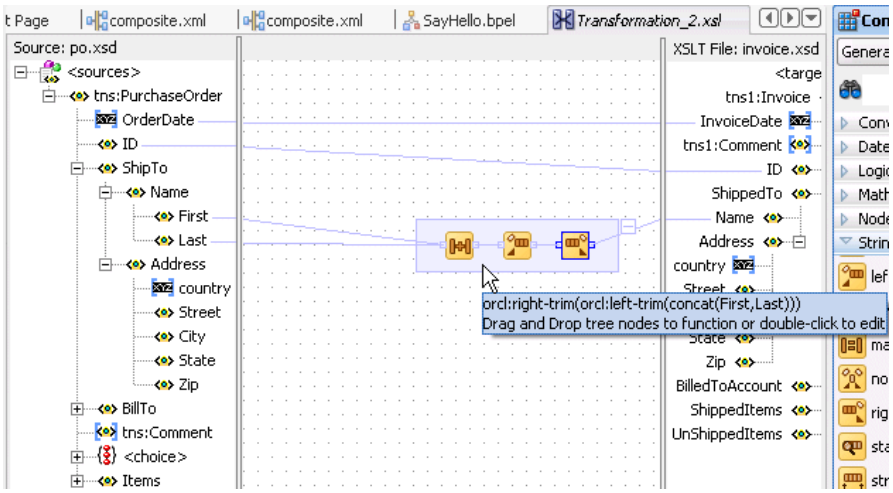
Complex expressions can be built by chaining functions (that is, mapping the output of one function to the input of another). For example, to remove all leading and trailing spaces from the output of the **concat** function, perform the following steps:

1. Drag the left-trim and right-trim functions into the border area of the **concat** function.

2. Chain them as shown in Figure 37–20 by dragging lines from the output side of one function to the input side of the next function.

Chaining can also be performed by dragging and dropping a function onto a connecting link.

Figure 37–20 Chaining Functions



37.3.4.3 Using Named Templates

Some complicated mapping logic cannot be represented or achieved by visual mappings. For these situations, named templates are useful. Named templates enable you to share common mapping logic. You can define the common mapping logic as a named template and then use it as often as you want.

You can define named templates in two ways:

- Add the template directly to your XSL map in source view.
- Add the template to an external file that you include in your XSL map.

The templates you define appear in the **User Defined Named Templates** list of the **User Defined** page in the Component Palette. You can use named templates in almost the same way as you use other functions. The only difference is that you cannot link the output of a named template to a function or another named template; you can only link its output to a target node in the target tree.

To create named templates, you must be familiar with the XSLT language. See any XSLT book or visit the following URL for details about writing named templates:

`http://www.w3.org/TR/xslt`

For more information about including templates defined in external files, see Section 37.3.6.7, "Including External Templates with xsl:include."

37.3.4.4 Importing User-Defined Functions

You can create and import a user-defined Java function if you have complex functionality that cannot be performed in XSLT or with XPath expressions.

Follow these steps to create and use your own functions. External, user-defined functions can be necessary when logic is too complex to perform within the XSL map.

To import user-defined functions:

1. Code and build your functions.

   The XSLT Mapper extension functions are coded differently than the Oracle BPEL Process Manager extension functions. Two examples are provided in the `SampleExtensionFunctions.java` file of the `mapper-107-extension-functions` sample scenario. Example 37–6 provides the text for these functions. To download these and other samples, visit the following URL:

   `https://soasamples.samplecode.oracle.com/`

   Each function must be declared as a static function. Input parameters and the returned value must be declared as one of the following types:

   - `java.lang.String`
   - `int`
   - `float`
   - `double`
   - `boolean`
   - `oracle.xml.parser.v2.XMLNodeList`
   - `oracle.xml.parser.v2.XMLDocumentFragment`

Example 37–6 XSLT Mapper Extension Functions

```
// SampleExtensionFunctions.java
package oracle.sample;
/*
This is a sample XSLT Mapper User Defined Extension Functions implementation
class.
*/
public class SampleExtensionFunctions
{
   public static Double toKilograms(Double lb)
   {
      return new Double(lb.doubleValue()*0.45359237);
   }
   public static String replaceChar(String inputString, String oldChar, String
      newChar )
   {
      return inputString.replace(oldChar.charAt(0), newChar.charAt(0));
   }
}
```

2. Create an XML extension function configuration file. This file defines the functions and their parameters.

   This file must have the name `ext-mapper-xpath-functions-config.xml`. See Section B.7, "Creating User-Defined XPath Extension Functions" for more information on the format of this file. The file shown in Example 37–7 represents the functions `toKilograms` and `replaceChar` as they are coded in Example 37–6.

Example 37–7 XML Extension Function Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<soa-xpath-functions version="11.1.1"
```

```
 xmlns="http://xmlns.oracle.com/soa/config/xpath" xmlns:sample=
"http://www.oracle.com/XSL/Transform/java/oracle.sample.SampleExtensionFunctions"
 >
      <function name="sample:toKilograms">
            <className>oracle.sample.SampleExtensionFunctions</className>
            <return type="number"/>
            <params>
                  <param name="pounds" type="number"/>
            </params>
            <desc>Converts a value in pounds to kilograms</desc>
      </function>
      <function name="sample:replaceChar">
            <className>oracle.sample.SampleExtensionFunctions</className>
            <return type="string"/>
            <params>
                  <param name="inputString" type="string"/>
                  <param name="oldChar" type="string"/>
                  <param name="newChar" type="string"/>
            </params>
            <desc>Returns a new string resulting from replacing all occurrences
                  of oldChar in this string with newChar</desc>
      </function>
</soa-xpath-functions>
```

Some additional rules apply to the definitions of XSLT extension functions:

- The functions need a namespace prefix and a namespace. In this sample, they are `sample` and `http://www.oracle.com/XSL/Transform/java/oracle.sample.SampleExtensionFunctions`.
- The function namespace must start with `http://www.oracle.com/XSL/Transform/java/` for extension functions to work with the Oracle XSLT processor.
- The last portion of the namespace, in this sample `oracle.sample.SampleExtensionFunctions`, must be the fully qualified name of the Java class that implements the extension functions.
- The types and their equivalent Java types shown in Table 37–1 can be used for parameter and return values:

Table 37–1 Types and Equivalent Java Types

XML Configuration File Type Name	Java Type
`string`	`java.lang.String`
`boolean`	`boolean`
`number`	`int`, `float`, `double`
`node-set`	`oracle.xml.parser.v2.XMLNodeList`
`tree`	`oracle.xml.parser.v2.XMLDocumentFragment`

3. Create a JAR file containing both the XML configuration file and the compiled classes. The configuration file must be contained in the `META-INF` directory for the JAR file. For the example in this section, the directory structure is as follows with the `oracle` and `META-INF` directories added to a JAR file:
   - `oracle`

  - `sample` (contains the class file)
- `META-INF`
  - `ext-mapper-xpath-functions-config.xml`

The JAR file must then be registered with Oracle JDeveloper.

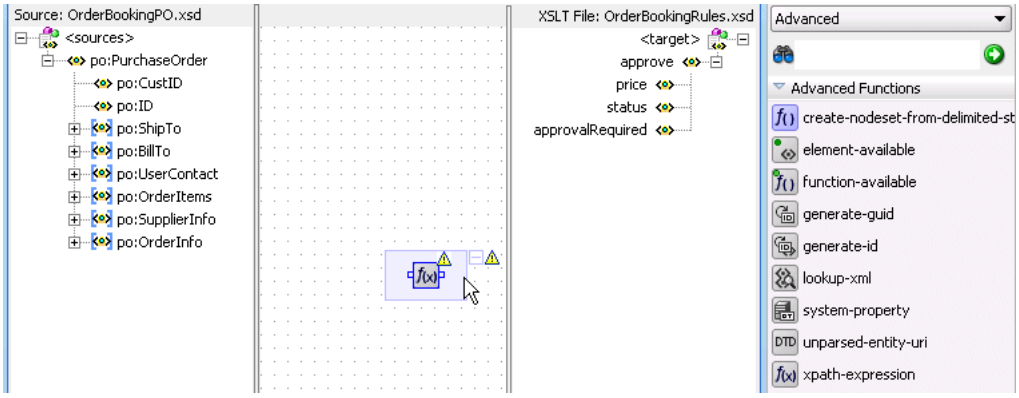
4. Go to **Tools** > **Preferences** > **SOA**.
5. Click the **Add** button and navigate to and select your JAR file.
6. Restart Oracle JDeveloper.

   New functions appear in the Component Palette under the **User Defined** page in the **User Defined Extension Functions** group.
7. To make the functions available in the runtime environment, Section B.7.3, "How to Deploy User-Defined Functions to Runtime" for details.

37.3.5 How to Edit XPath Expressions

To use an XPath expression in a transformation mapping, select the **Advanced** page and then the **Advanced Function** group from the Component Palette and drag **xpath-expression** from the list into the XSLT Mapper. This is shown in Figure 37–21.

Figure 37–21 Editing XPath Expressions



When you double-click the icon, the Edit XPath Expression dialog appears, as shown in Figure 37–22. You can press Ctrl+Space to invoke the XPath Building Assistant.

Figure 37–22 Edit XPath Expression Dialog

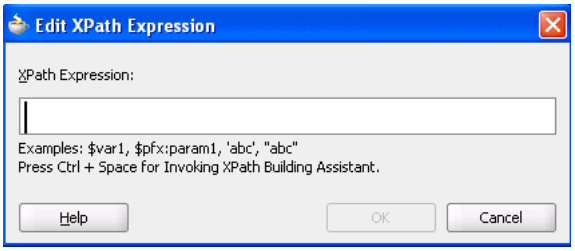
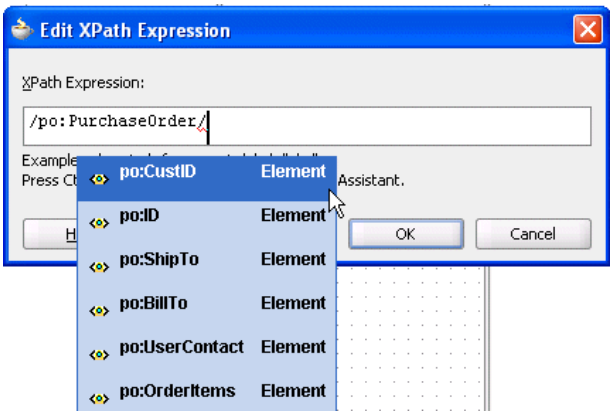


Figure 37–23 shows the XPath Building Assistant.

Figure 37–23 The XPath Building Assistant



For more information about using the XPath Building Assistant, see the online Help for the Edit XPath Expression dialog and Section B.6, "Building XPath Expressions in Oracle JDeveloper."

37.3.6 How to Add XSLT Constructs

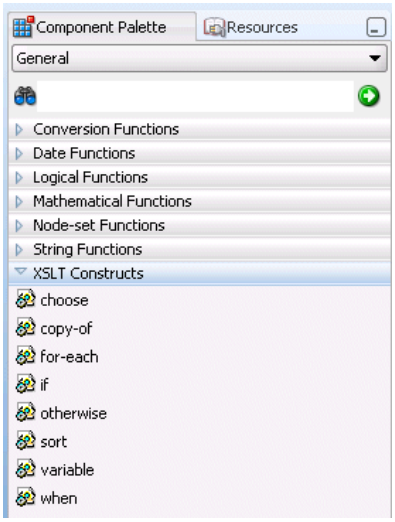
While mapping complex schemas, it is essential to be able to add XSLT constructs. For instance, you may need to create a node in the target when a particular condition exists; this requires the use of an `xsl:if` statement or an `xsl:choose` statement. You may also need to loop over a `node-set` in the source such as a list of items in a sales order and create nodes in the target XML for each item in the sales order; this requires the use of an `xsl:for-each` statement. The XSLT Mapper provides XSLT constructs for performing these and other tasks.

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

To add XSLT constructs from the Component Palette:

1. Select the **General** page and open the **XSLT Constructs** group. Figure 37–24 provides details.

Figure 37–24 XSLT Constructs Available Through the Component Palette

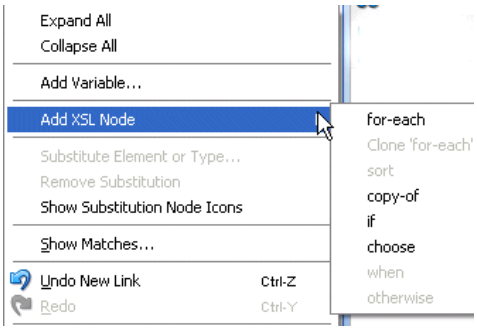


2. Drag an XSLT construct from the group onto a node in the target tree. If the XSLT construct can be applied to the node, it is inserted in the target tree. Note that the **when** and **otherwise** constructs must be applied to a previously-inserted **choose** node.

To add XSLT constructs through the context menu on the target tree:

1. Right-click the element in the target tree where you want to insert an XSLT construct. A context menu is displayed. Figure 37–25 provides details.

Figure 37–25 XSLT Constructs in Available Through the Context Menu



2. Select **Add XSL Node** and then the XSLT construct you want to insert.

The XSLT construct is inserted. In most cases, an error icon initially appears next to the construct. This indicates that the construct requires an XPath expression to be defined for it.

In the case of the **for-each** construct, for example, an XPath expression defines the node set over which the **for-each** statement loops. In the case of the **if** construct, the XPath expression defines a boolean expression that is evaluated to determine if the contents of the **if** construct are executed.

The XPath expression can be created in the same manner as mapping elements and attributes in the target tree. The following methods create an underlying XPath expression in the XSLT. You can perform all of these methods on XSLT constructs in the target tree to set their XPath expressions:

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

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

37.3.6.1 Using Conditional Processing with xsl:if

In Figure 37–26, note that **HQAccount** and **BranchAccount** are part of a choice in the **PurchaseOrder** schema; only one of them exists in an actual instance. To illustrate conditional mapping, copy **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/AccountNumber**, only if it exists.

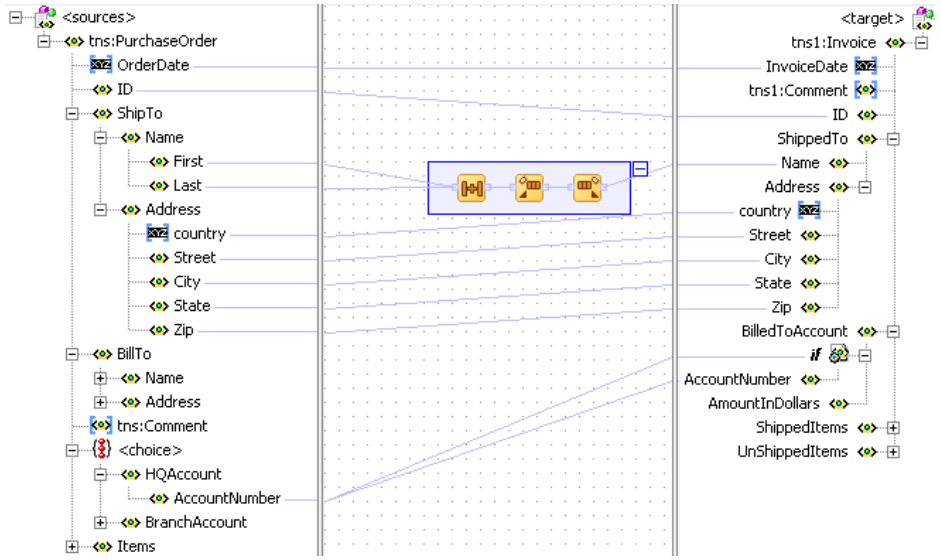
To use conditional processing with xsl:if:

1. In the target tree, select **Invoice/BilledToAccount/AccountNumber** and right-click to invoke the context sensitive menu.

2. Select **Add XSL Node** > **if** and connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/if**.
3. Connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/if/AccountNumber**.

Figure 37–26 shows the results.

Figure 37–26 Conditional Processing with xsl:if



When mapping an optional source node to an optional target node, it is important to surround the mapping with an `xsl:if` statement that tests for the existence of the source node. If this is not done and the source node does not exist in the input document, an empty node is created in the target document. For example, note the mapping shown in Example 37–8:

Example 37–8 Statement Without xsl:If

```
<ProductName>
  <xsl:value-of select="ProductName"/>
</ProductName>
```

If the `ProductName` field is optional in both the source and target and the element does not exist in the source document, then an empty `ProductName` element is created in the target document. To avoid this situation, add an `if` statement to test for the existence of the source node before the target node is created, as shown in Example 37–9:

Example 37–9 Statement With xsl:If

```
<xsl:if test="ProductName">
    <ProductName>
      <xsl:value-of select="ProductName"/>
    </ProductName>
</xsl:if>
```

37.3.6.2 Using Conditional Processing with xsl:choose

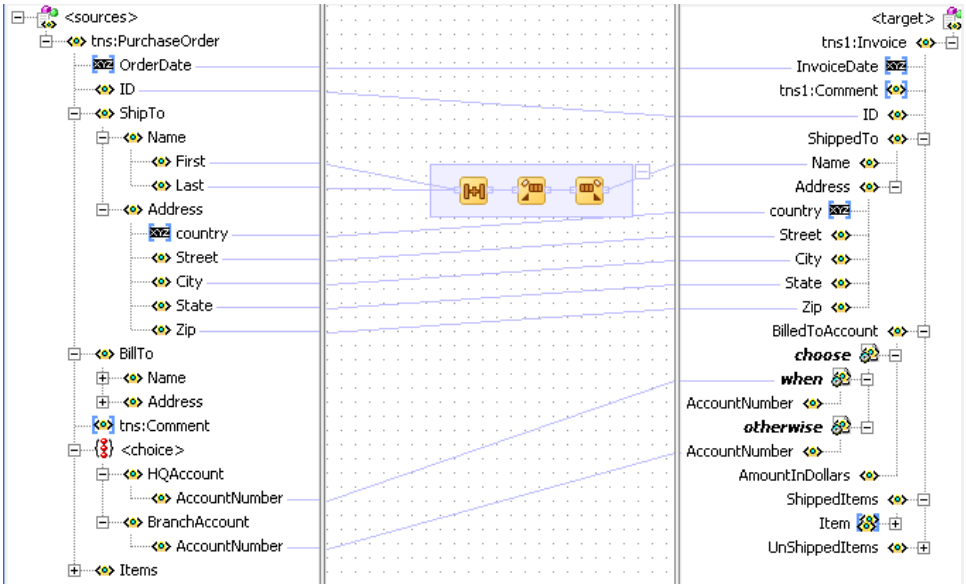
In this same example, you can copy **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/AccountNumber**, if it exists. Otherwise, copy **PurchaseOrder/BranchAccount** to **Invoice/BilledToAccount/AccountNumber**.

To use conditional processing with xsl:choose:

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

Figure 37–27 shows the results.

Figure 37–27 Conditional Processing with xsl:choose



37.3.6.3 Creating Loops with xsl:for-each

The XSLT Mapper enables you to create loops with the **xsl:for-each** command. For example, copy **PurchaseOrder/Items/HighPriorityItems/Item** to **Invoice/ShippedItems/Item**.

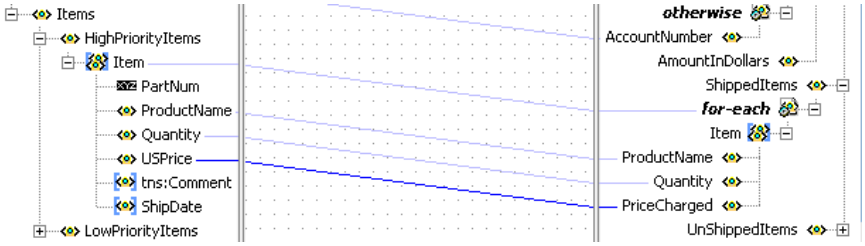
To create loops with xsl:for-each:

1. In the target tree, select **Invoice/ShippedItems/Item** and right-click to invoke the context sensitive menu.

2. Select **Add XSL Node** > **for-each** and connect **PurchaseOrder/Items/HighPriorityItems/Item** to **Invoice/ShippedItems/for-each** to define the iteration.
3. Connect **PurchaseOrder/Items/HighPriorityItems/Item/ProductName** to **Invoice/ShippedItems/for-each/Item/ProductName**.
4. Connect **PurchaseOrder/Items/HighPriorityItems/Item/Quantity** to **Invoice/ShippedItems/for-each/Item/Quantity**.
5. Connect **PurchaseOrder/Items/HighPriorityItems/Item/USPrice** to **Invoice/ShippedItems/for-each/Item/PriceCharged**.

Figure 37–28 shows the results.

Figure 37–28 Creating Loops with xsl:for-each



Notes:

- Executing an auto map automatically inserts **xsl:for-each.** To see the auto map in use, drag **PurchaseOrder/Items/LowPriorityItems** to **Invoice/UnShippedItems**; **for-each** is automatically created.
- Ensure that your design does not include infinite loops. These loops result in errors similar to the following displaying during deployment and invocation of your application.

```
ORAMED-04001:
. . .
oracle.tip.mediator.service.BaseActionHandler requestProcess
SEVERE:
failed reference BPELProcess1.bpelprocess1_client operation =
process
```

37.3.6.4 Cloning xsl:for-each

You can create additional loops by cloning an existing **xsl:for-each**. For example, copy all **LowPriorityItems** to **ShippedItems**, in addition to **HighPriorityItems**.

To clone xsl:for-each:

1. Under **Invoice/ShippedItems**, select **for-each**.
2. Right-click and select **Add XSL Node** > **Clone 'for-each'**.

   This inserts a copy of the **for-each** node beneath the original **for-each**.
3. Drag **PurchaseOrder/Items/LowPriorityItems/Item** to the copied **for-each** to define the iteration.
4. Connect **PurchaseOrder/Items/LowPriorityItems/Item/ProductName** to **Item/ProductName** in the copied **for-each**.

5. Connect **PurchaseOrder/Items/LowPriorityItems/Item/Quantity** to **Item/Quantity** in the copied **for-each**.
6. Connect **PurchaseOrder/Items/LowPriorityItems/Item/USPrice** to **Item/PriceCharged** in the copied **for-each**.

37.3.6.5 Applying xsl:sort to xsl:for-each

The XSLT Mapper enables you to add **xsl:sort** statements to **xsl:for-each** commands.

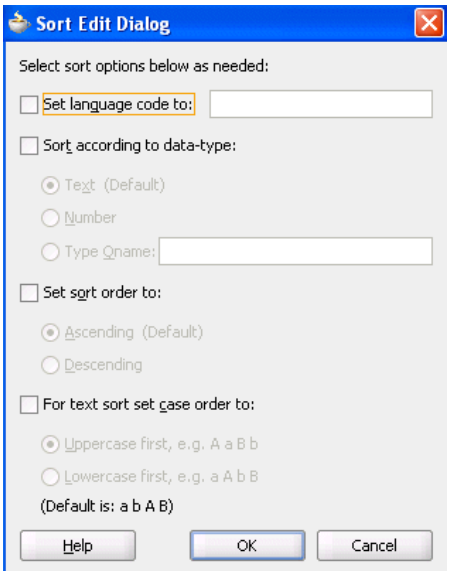
To add an xsl:sort statement:

1. Right-click a **for-each** statement in the target tree.

   A context menu appears.

2. Select **Add XSL Node** > **sort**. The Sort Edit Dialog is displayed, as shown in Figure 37–29.

Figure 37–29 Sort Edit Dialog



3. Select options to add to the sort statement as needed. See the online Help for information on options.
4. Click **OK**. The sort statement is added following the **for-each**.
5. To set the field on which to sort, drag the necessary sort field node in the source tree to the sort node in the target tree. This creates a simple link and sets the XPath expression for the select attribute on the **xsl:sort**.
6. To add additional sort statements, right-click the **for-each** to add another sort or right-click an existing sort node to insert a new sort statement before the selected sort node.
7. To edit a sort node, double-click the sort node or right-click and select **Edit Sort** from the context menu. This invokes the Sort Edit Dialog and enables you to change the sort options.

37.3.6.6 Copying Nodes with xsl:copy-of

You may need to use the XSLT **copy-of** construct to copy a node, along with any child nodes, from the source to the target tree. This is typically done when working with

anyType or **any** element nodes. Note that **anyType** and **any** element and attribute nodes cannot be mapped directly. Use **copy-of** or element and type substitution.

To copy nodes with xsl:copy-of:

1. Select the node in the target tree to be created by the **copy-of** command.
2. Right-click the node and select **Add XSL Node** > **copy-of**.

   If the node is not an **any** element node, a dialog appears requesting you to either replace the selected node or replace the children of the selected node.
3. Select the correct option for your application and click **OK**.

   If you select **Replace the selected node** with the **copy-of**, a processing directive is created immediately following the **copy-of** in the XSL indicating which node is replaced by the **copy-of**. Without the processing directive in the XSL, the conversion back to design view is interpreted incorrectly. For this reason, do not remove or edit this processing instruction while in source view.
4. Set the source node for the **copy-of** by dragging and dropping from the source tree or by creating an XPath expression.

Note: Always create the **copy-of** command in design view so that the correct processing directive can be created in the XSLT Mapper to indicate the correct placement of the **copy-of** command in the target tree.

WARNING: The XSLT Mapper does not currently validate the mapping of data performed through use of the copy-of command. You must ensure that copy-of is used to correctly map elements to the target tree so that the target XML document contains valid data. You can test the validity by using the test tool.

37.3.6.7 Including External Templates with xsl:include

You can reuse templates that are defined in external XSL files by including them in the current map with an include statement.

To include external templates with xsl:include:

1. In the target tree, select and right-click the root node.
2. From the menu, select **Add Include File**.

   A dialog prompts you for the include file name.
3. Select the file and click **OK**.

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

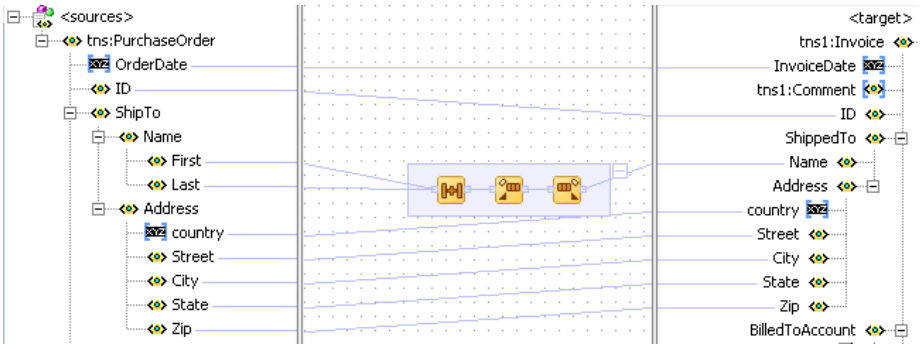
   The include file can only contain named template definitions. These are parsed and available to you in design view of the Component Palette under the **User Defined Named Templates** category in the **User Defined** page.

Note: An `oramds://` shared location cannot be referenced for a user-defined named template include file.

37.3.7 How to Automatically Map Nodes

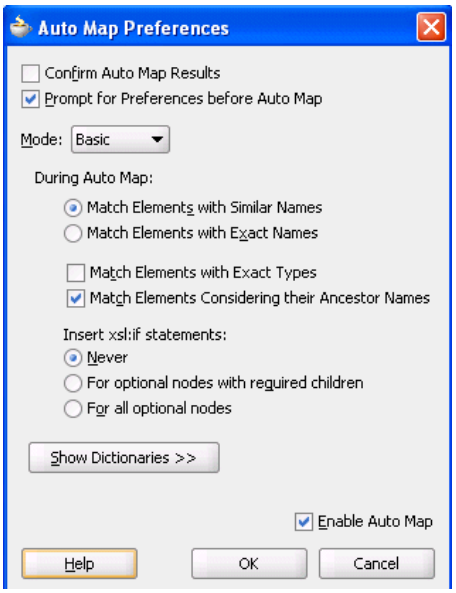
Mapping nonleaf nodes starts the auto map feature. The system automatically tries to link all relevant nodes under the selected source and target. Try the auto map feature by mapping **PurchaseOrder/ShipTo/Address** to **Invoice/ShippedTo/Address**. All nodes under **Address** are automatically mapped, as shown in Figure 37–30.

Figure 37–30 Auto Mapping



The behavior of the auto map can be tuned by altering the settings in Oracle JDeveloper preferences or by right-clicking the XSLT Mapper and selecting **Auto Map Preferences**. This displays the dialog shown in Figure 37–31.

Figure 37–31 Auto Map Preferences



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

- Invoke the automatic mapping feature, which attempts to automatically link all relevant nodes under the selected source and target. When disabled, you must individually map relevant nodes.

- Display and review all potential source-to-target mappings detected by the XSLT Mapper, and then confirm to create them.
- Be prompted to customize the auto map preferences before the auto map is invoked.
- Select the **Basic** or **Advanced** method for automatically mapping source and target nodes. This action enables you to customize how the XSLT Mapper attempts to automatically link all relevant nodes under the selected source and target.
- Manage your dictionaries. The XSLT Mapper uses the rules defined in a dictionary when attempting to automatically map source and target elements.

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

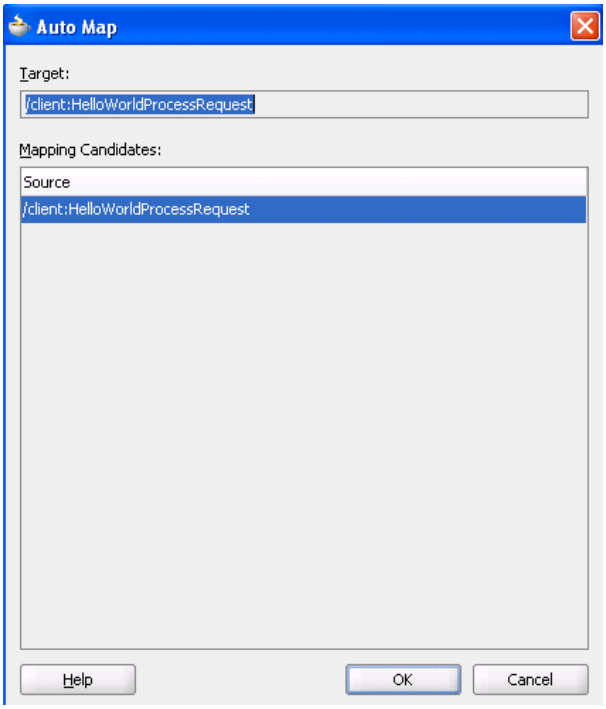
Follow these instructions to see potential source mapping candidates for a target node.

To automatically map nodes:

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

   The Auto Map dialog appears, as shown in Figure 37–32.

Figure 37–32 Auto Mapping Candidates

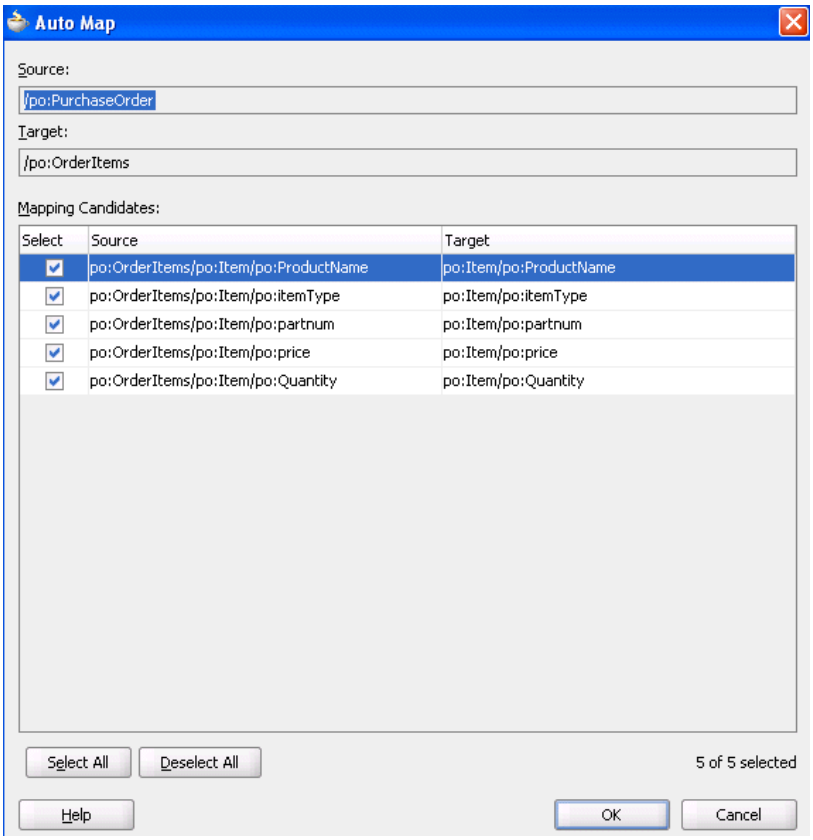


For more information on the fields, see the online Help for the Auto Map dialog.

37.3.7.1 Using Auto Mapping with Confirmation

When the **Confirm Auto Map Results** checkbox shown in Figure 37–31 is selected, a confirmation dialog appears. If matches are found, the potential source-to-target mappings detected by the XSLT Mapper are displayed, as shown in Figure 37–33. The dialog enables you to filter one or more mappings.

Figure 37–33 Auto Map with Confirmation



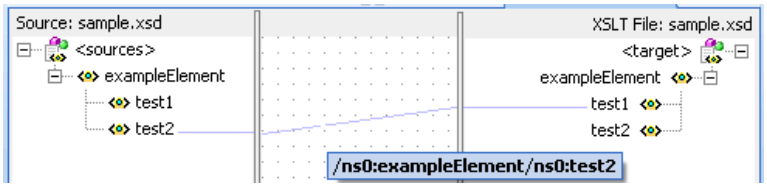
For more information about the fields, see the online Help for the Auto Map dialog.

37.3.8 What You May Need to Know About Automatic Mapping

The automatic mapping algorithm depends on existing maps between source and target nodes. When maps exist between source and target nodes before executing automatic mapping, these existing maps are used to define valid synonyms that are used by the algorithm.

For example, assume you have a simple source and target tree, each with two elements called **test1** and **test2**, as shown in Figure 37–34.

Figure 37–34 Source and Target Tree with Two Elements

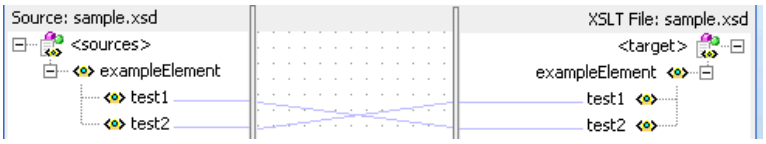


If no nodes are mapped, the automatic mapping algorithm does not match the names **test1** and **test2**. However, if mapping exists between the **test1** and **test2** nodes, the algorithm predefines the names **test1** and **test2** as synonyms for any additional mapping.

In the example in Figure 37–34, if you drag the **exampleElement** from the source to the target, the automatic mapping algorithm maps the **test1** node in the source to the **test2**

node in the target because your map previously linked those two names. This results in the map shown in Figure 37–35:

Figure 37–35 Results of Dragging exampleElement



37.3.9 How to View Unmapped Target Nodes

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

To view unmapped target nodes:

1. In the XSLT Mapper, right-click in the center panel and select **Completion Status**.

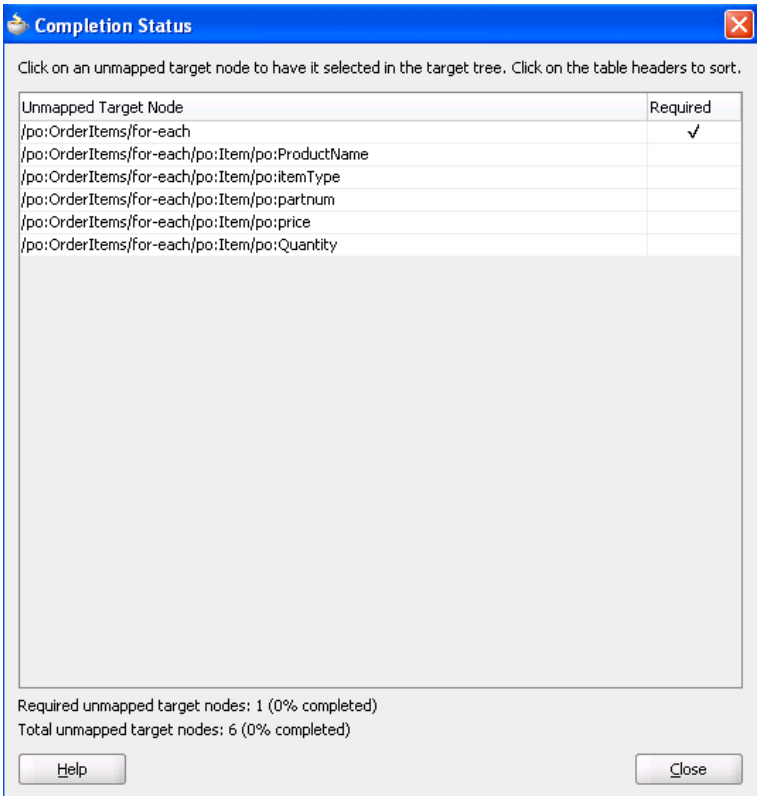
   This dialog provides statistics at the bottom about the number of unmapped target nodes. This dialog enables you to identify and correct any unmapped nodes before you test your transformation mapping logic on the Test XSL Map dialog.

2. In the list, select a target node. The node is highlighted. A checkmark indicates that the target node is required to be mapped. If not required, the checkbox is empty.

   Note: Nodes are marked as required in the Completion Status dialog based on the XSD definition for a node. It is possible that a node marked as required is not actually required for a specific mapping if a parent node of the required node is optional and is not part of the XSL mapping.

   Figure 37–36 provides an example of the Completion Status dialog.

Figure 37–36 Completion Status



37.3.10 How to Generate Dictionaries

A dictionary is an XML file used by automatic mapping. It contains synonyms for field names. For instance, assume that the element **QtyOrdered** should map to the element **Quantity**. The element names **QtyOrdered** and **Quantity** are then synonyms for one another. If this mapping commonly appears from one map to another, it is a good practice to save these synonyms in a dictionary file. After being saved, they can be reapplied to another map using automatic mapping.

A dictionary can be created from any existing XSL map and can contain all mappings that are not automatically generated by the XSLT Mapper for the existing map.

To generate and use dictionaries:

1. Create an XSL map that contains specific mappings to reuse in other maps.
2. Go to **Tools** > **Preferences** > **XSL Maps** > **Auto Map** and note the current automatic mapping settings.

   Note: Because dictionary entries are dependent upon the current automatic mapping settings, you must make a note of those settings for future use. To later reapply a dictionary mapping, it is best to set the automatic mapping preferences to those that were in effect at the time the dictionary was created. Therefore, it is important to note the automatic mapping settings at the time the dictionary is created.

3. In the XSLT Mapper, right-click the center panel of the XSLT Mapper and select **Generate Dictionary**.

This prompts you for the dictionary name and the directory in which to place the dictionary.

4. Check the **Open Dictionary** checkbox to view the dictionary after it is created. If the dictionary file is empty, this indicates that no fields were mapped that would not have been mapped with the current automatic mapping settings.
5. To use the dictionary in another map, load the dictionary by selecting **Tools** > **Preferences** > **XSL Maps** > **Auto Map**.
6. Click **Add** below the **Dictionaries** list.
7. Browse for and select the dictionary XML file that was previously created from a similar map.
8. Click **OK**.
9. Before leaving the automatic mapping preferences, modify the mapping settings to match those used when creating the dictionary.
10. Click **OK**.
11. Perform an automatic mapping of whichever portion of the new map corresponds to the saved dictionary mappings.

For more information about automatic mapping, see Section 37.3.7, "How to Automatically Map Nodes."

37.3.11 What You May Need to Know About Generating Dictionaries in Which Functions are Used

You cannot create a dictionary for mappings in which functions are used. In these cases, the dictionary XML instructions are missing for the elements that were automatically mapped or which had an XPath function mapping. For example, assume you use string functions to map XSDs during design time. If you right-click the center panel of the XSLT Mapper and select **Generate Dictionary**, the dictionary is created, but instructions are not created in all places in which the string functions were used during mapping.

Note that you can create a dictionary for simple type mappings.

37.3.12 How to Create Map Parameters and Variables

You can create map parameters and variables. You create map parameters in the source tree and map variables in the target tree.

Note the following issues:

- Parameters are created in the source tree, are global, and can be used anywhere in the mappings.
- Variables are created in the target tree, and are either global or local. The location in which they are defined in the target tree determines if they are global or local.
  - Global variables are defined immediately beneath the **<target>** node and immediately above the actual target schema (for example, **POAcknowledge**). Right-click the **<target>** node to create a global variable.
  - Local variables are defined on a specific node beneath the actual target schema (for example, subnode **name** on schema **POAcknowledge**). Local variables can have the same name provided they are in different scopes. Local variables

can only be used in their scopes, while global variables can be used anywhere in the mappings.

37.3.12.1 Creating a Map Parameter

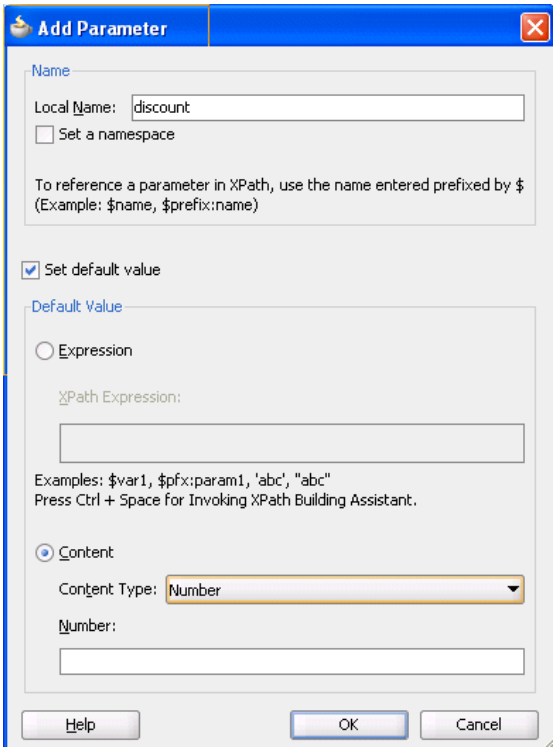
To create a map parameter:

1. In the source tree root, right-click and select **Add Parameter**.

   The Add Parameter dialog shown in Figure 37–37 appears.

2. Specify details for the parameter. For this example, a parameter named `discount` with a numeric default value of `0.0` is specified.

Figure 37–37 Add Parameter Dialog



3. Click **OK**.

37.3.12.2 Creating a Map Variable

To create a map variable:

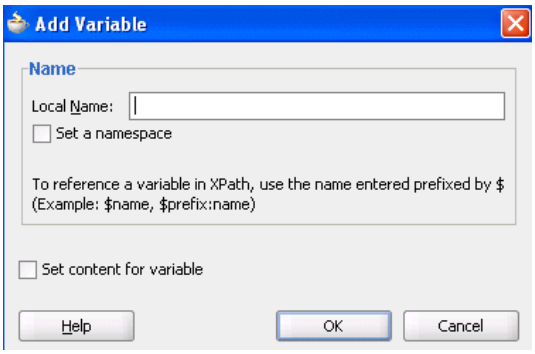
1. In the target tree, right-click the target tree root or any node and select **Add Variable**.

   The Add Variable dialog shown in Figure 37–38 appears.

2. Specify details.

   Since variables appear in the target tree, their XPath expression can be set in the same manner as other XSLT constructs in the target tree after inserting the variable. Therefore, the only required information in this dialog is a name for the variable. To set content for the variable, you must enter it through this dialog. Content is handled differently from the XSLT `select` attribute on the variable.

Figure 37–38 Add Variable Dialog



3. Click **OK**.

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

   The variable initially has a warning icon beside it. This indicates that its select XPath statement is undefined. Define the XPath through linking a source node, creating a function, or defining an explicit XPath expression as done for other target elements and XSLT constructs.

37.3.13 How to Search Source and Target Nodes

You can search source and target nodes. For example, you can search in a source node named **invoice** for all occurrences of the subnode named **price**.

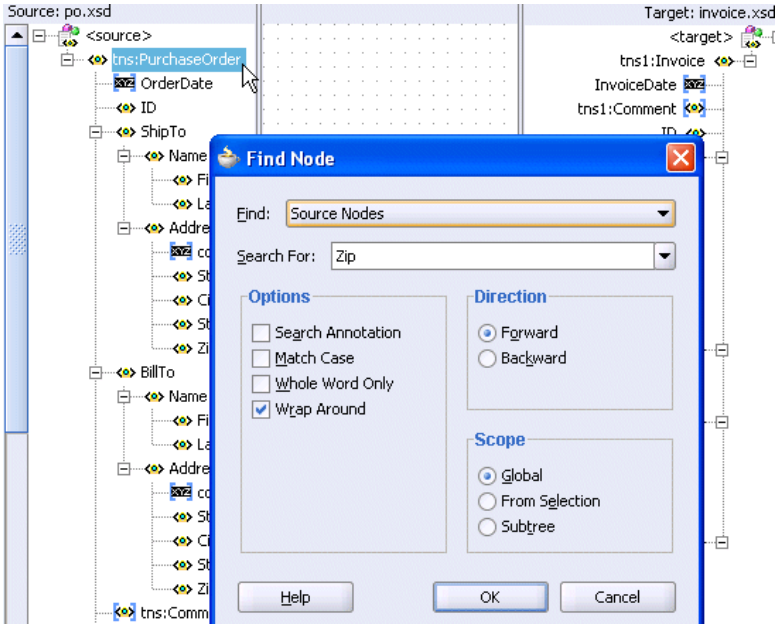
To search source and target nodes:

1. Right-click a source or target node and select **Find** from the context menu.

   The Find Node dialog shown in Figure 37–39 is displayed.

2. Enter a keyword for which to search.
3. Specify additional details, as necessary. For example:
   - Select **Search Annotations** if you want annotations text to also be searched.
   - Specify the scope of the search. You can search the entire source or target tree, search starting from a selected position, or search within a selected subtree.

Figure 37–39 Find Node Dialog



The first match found is highlighted, and the Find dialog closes. If no matches are found, a message displays on-screen.

4. Select the F3 key to find the next match in the direction specified. To search in the opposite direction, select the Shift and F3 keys.

Note: You cannot search on functions or text values set with the **Set Text** option.

37.3.14 How to Control the Generation of Unmapped Target Elements

There are five options for controlling the generation of empty elements in the target XSL:

- Do not generate unmapped nodes (default option).
- Generate empty nodes for *all* unmapped target nodes.
- Generate empty nodes for *all* required, unmapped target nodes.
- Generate empty nodes for *all* nillable, unmapped target nodes.
- Generate empty nodes for *all* required or nillable, unmapped target nodes.

Set these options as follows:

- At the global level:

  Select **Tools** > **Preferences** > **XSL Maps**. The global setting applies only when a map is created.

- At the map level:

  Select **XSL Generation Options** from the map context menu. Each map can then be set independently by setting the options at the map level.

Empty elements are then generated for the selected unmapped nodes. If the unmapped node is nillable, it is generated with `xsi:nil="true"`.

37.3.15 How to Ignore Elements in the XSLT Document

When the XSLT Mapper encounters any elements in the XSLT document that cannot be found in the source or target schema, it cannot process them and displays an `Invalid Source Node Path` error. XSL map generation fails. You can create and import a file that directs the XSLT Mapper to ignore and preserve these specific elements during XSLT parsing by selecting **Preferences** > **XSL Maps** in the **Tools** main menu of Oracle JDeveloper.

For example, preprocessing may create elements named `myElement` and `myOtherElementWithNS` that you want the XSLT Mapper to ignore when it creates the graphical representation of the XSLT document. You create and import a file with these elements to ignore that includes the syntax shown in Example 37–10.

Example 37–10 File with Elements to Ignore

```
<elements-to-ignore>
   <element name="myElement"/>
   <element name="myOtherElementWithNS" namespace="NS"/>
</elements-to-ignore>
```

You must restart Oracle JDeveloper after importing the file.

37.3.16 How to Replace a Schema in the XSLT Mapper

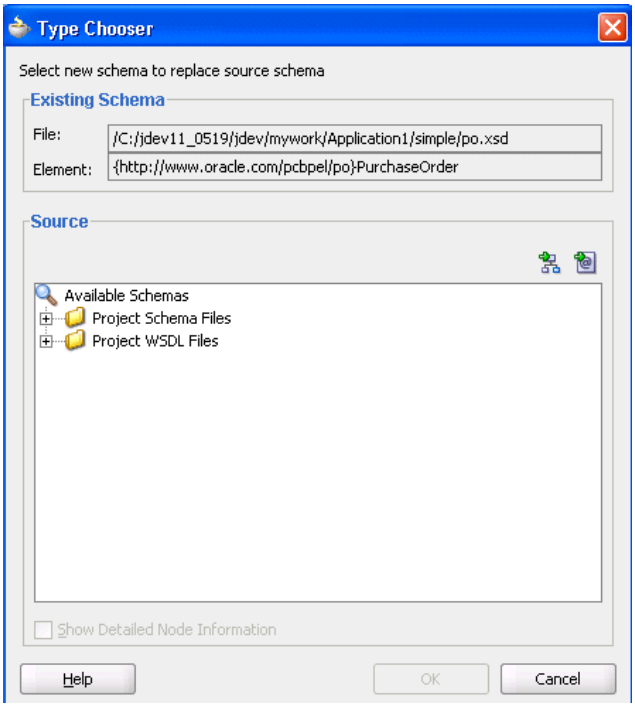
You can replace the map source or target schema that currently displays in the XSLT Mapper.

To replace a schema in the XSLT Mapper:

1. In either the source or target panel, right-click and select **Replace Schema**.

   This opens the Type Chooser dialog shown in Figure 37–40, which enables you to select the new source or target schema to use.

Figure 37–40 Replacing a Schema



2. Select the replacement schema and click **OK**.

   You are then prompted to select to clear expressions in the current map.

3. Select **Yes** or **No**. If expressions are not cleared, you may need to correct the map in source view before entering design view again.

37.3.17 How to Substitute Elements and Types in the Source and Target Trees

You can substitute elements and types in the source and target trees.

Use element substitution when:

- An element is defined as the head of a substitution group in the underlying schema. The element may or may not be abstract. Any element from the substitution group can be substituted for the original element.
- An element is defined as an `any` element. Any global element defined in the schema can be substituted.

Use type substitution when:

- A global type is available in the underlying schema that is derived from the type of an element in the source or target tree. The global type can then be substituted for the original type of the element. Any type derived from an abstract type can be substituted for that abstract type.
- An element in the source or target tree is defined to be of the type `anyType`. Any global type defined in the schema can then be substituted.

Type substitution is supported by use of the `xsi:type` attribute in XML.

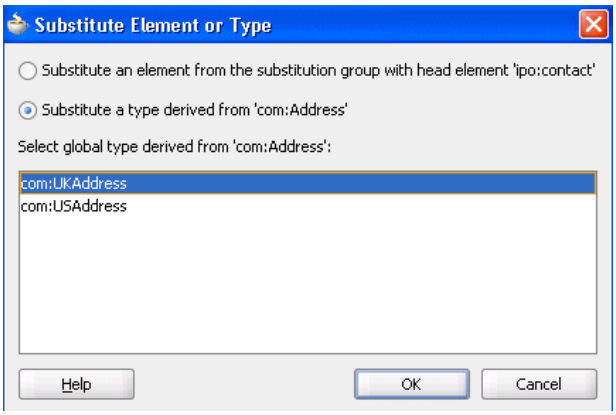
To substitute an element or type in the source and target trees:

1. In the source or target tree, right-click the element for which substitution applies.

2. From the context menu, select **Substitute Element or Type**. If this option is disabled, no possible substitutions exist for the element or its type in the underlying schema.

   The Substitute Element or Type dialog shown in Figure 37–41 appears.

Figure 37–41 Substitute Element or Type Dialog



3. Select either **Substitute an element** or **Substitute a type** (only one may be available depending upon the underlying schema).

   A list of global types or elements that can be substituted displays in the dialog.

4. Select the type or element to substitute.

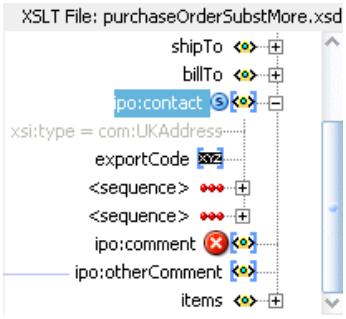
5. Click **OK**.

   The element or type is substituted for the originally selected element. This selection displays differently depending upon whether this is a type or element substitution and this is the source or target tree.

   - If the element is in the target tree and type substitution is selected:

     The **xsi:type** attribute is added beneath the original element, as shown in Figure 37–42. It is disabled in design view and set to the type value that was selected. An **S** icon displays to indicate the node was substituted. You can map to any structural elements in the substituted type.

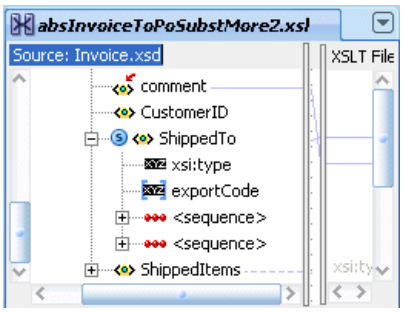
Figure 37–42 If the Element is in the Target Tree and Type Substitution is Selected



   - If the element is in the source tree and type substitution is selected:

     The **xsi:type** attribute is added beneath the original element, as shown in Figure 37–43. An **S** icon is displayed to indicate the node was substituted. You can map from any structural elements in the substituted type.

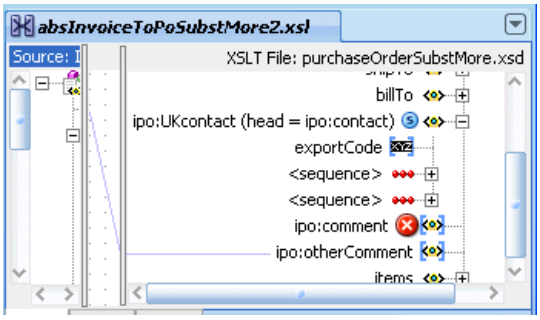
Figure 37–43 If the Element is in the Source Tree and Type Substitution is Selected



- If the element is in the target tree and element substitution is selected:

  The original element is replaced in the tree with the substituted element, as shown in Figure 37–44. A comment indicates that the original element name was added and an **S** icon displays to indicate the node was substituted. You may map to any structural elements in the substituted element.

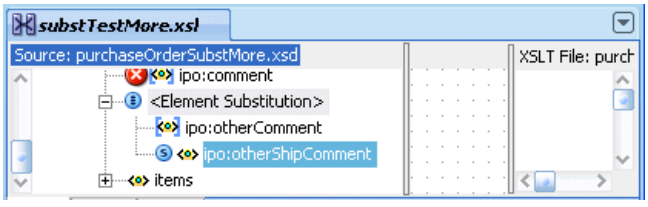
Figure 37–44 If the Element is in the Target Tree and Element Substitution is Selected



- If the element is in the source tree and element substitution is selected:

  The original element and its possible replacement both display in the source tree under a new node named **<Element Substitution>**, as shown in Figure 37–45. An **S** icon displays to indicate that the node was added. This feature enables you to build a map where the source object can contain either the original node or a substituted node. You can map to any structural elements in the substituted element.

Figure 37–45 If the Element is in the Source Tree and Element Substitution is Selected

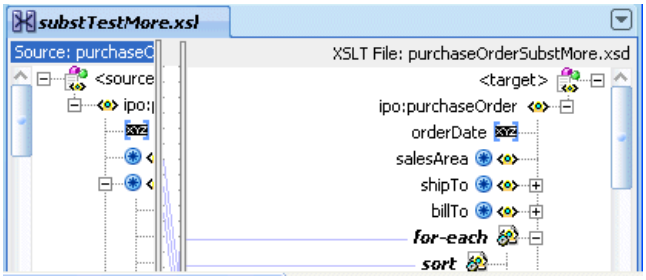


Note: Unlike element substitution, only one type substitution at a time can display in the source tree. This does not prevent you from writing a map that allows the source to sometimes have the original type or the substituted type; you can switch to another type at any time. XPath expressions that map to nodes that may not be visible in the source tree at any given time are still retained.

6. To remove a substituted node, right-click any node with an **S** icon and select **Remove Substitution** from the context menu.
7. To see all possible nodes where substitution is allowed, right-click the source or target tree and select **Show Substitution Node Icons**.

   All nodes where substitution is possible are marked with an * icon, as shown in Figure 37–46.

Figure 37–46 All Possible Substitutions

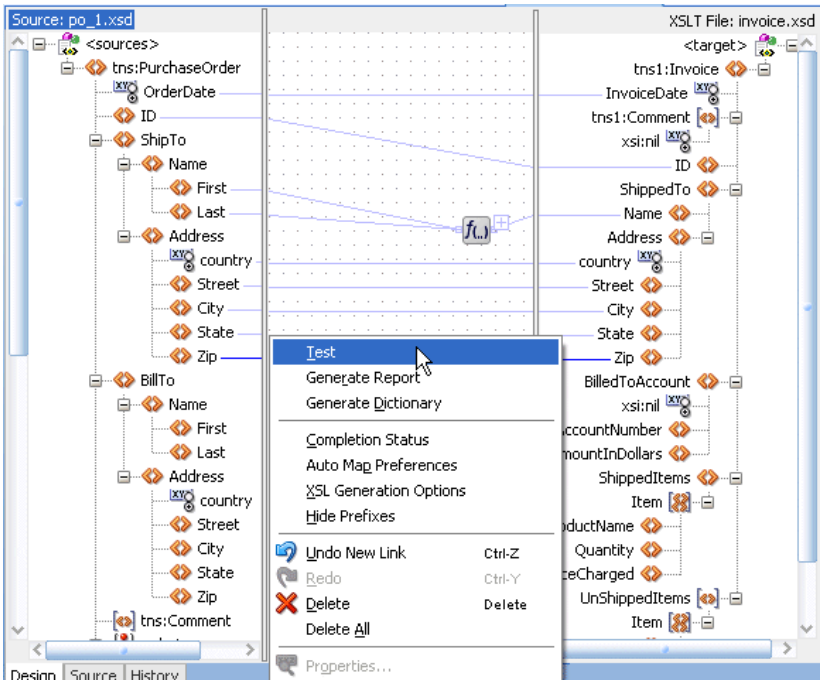


8. To hide the icons, right-click and select **Hide Substitution Node Icons**.

37.4 Testing the Map

The XSLT Mapper provides a test tool to test the style sheet or map. The test tool can be invoked by selecting the **Test** menu item, as shown in Figure 37–47.

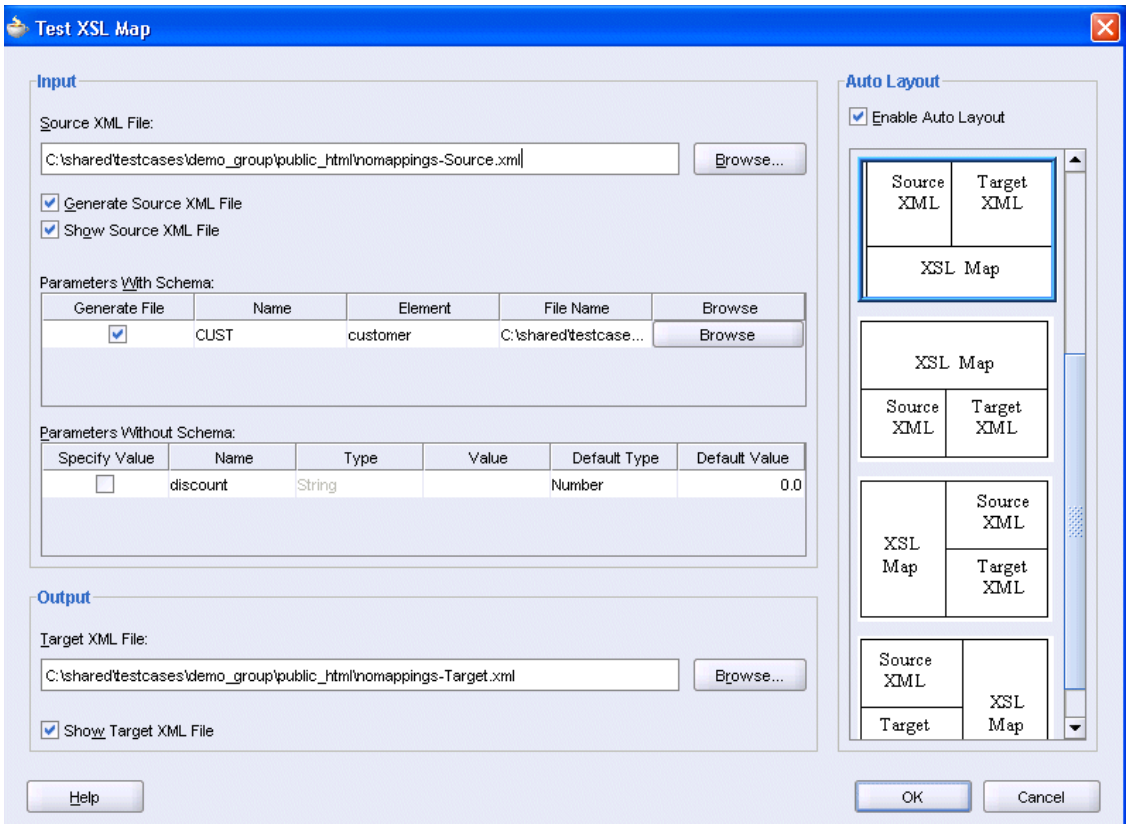
Figure 37–47 Invoking the Test Dialog



37.4.1 How to Test the Transformation Mapping Logic

The Test XSL Map dialog shown in Figure 37–48 enables you to test the transformation mapping logic you designed with the XSLT Mapper. The test settings you specify are stored and do not need to be entered again the next time you test. Test settings must be entered again if you close and reopen Oracle JDeveloper.

Figure 37–48 Test XSL Map Dialog



To test the transformation mapping logic:

1. In the **Source XML File** field, choose to allow a sample source XML file to be generated for testing or click **Browse** to specify a different source XML file.

   When you click **OK**, the source XML file is validated. If validation passes, transformation occurs, and the target XML file is created.

   If validation fails, no transformation occurs and a message displays on-screen.

2. Select the **Generate Source XML File** checkbox to create a sample XML file based on the map source XSD schema.

3. Select the **Show Source XML File** checkbox to display the source XML files for the test. The source XML files display in an Oracle JDeveloper XML editor.

   If the map has defined parameters, the **Parameters With Schema** or **Parameters Without Schema** table can appear.

   a. If the **Parameters With Schema** table appears, you can specify an input XML file for the parameter using the **Browse** button. Select the **Generate File** checkbox to generate a file.

   b. If the **Parameters Without Schema** table appears, you can specify a value by selecting the **Specify Value** checkbox and making appropriate edits to the **Type** and **Value** columns.

4. In the **Target XML File** field, enter a file name or browse for a file name in which to store the resulting XML document from the transformation.

5. Select the **Show Target XML File** checkbox to display the target XML file for the test. The target XML file displays in an Oracle JDeveloper XML editor.

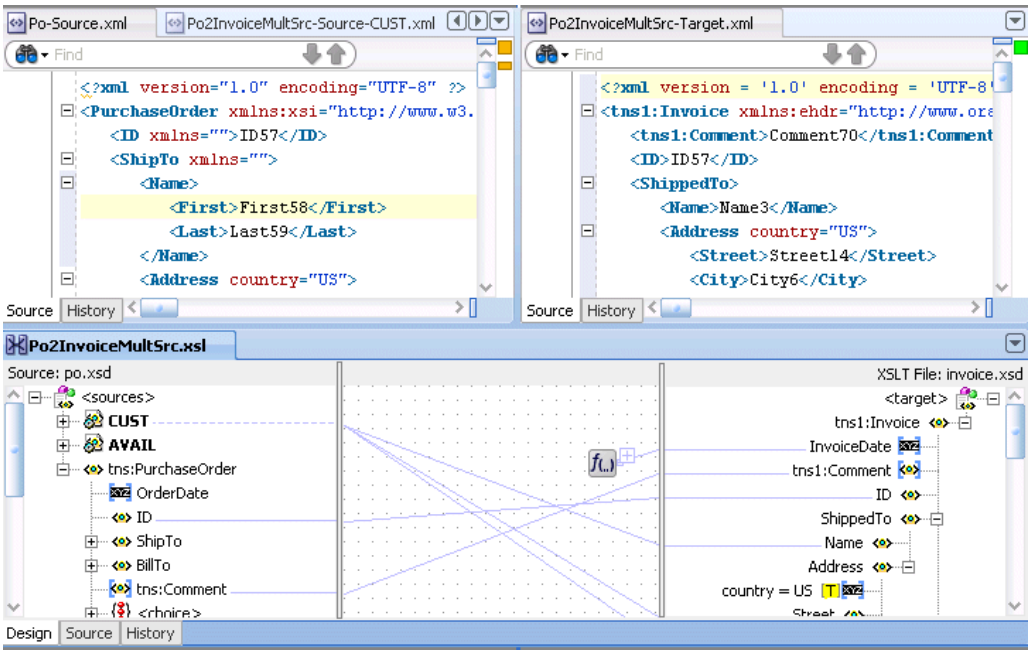
6. If you select to show both the source and target XML, you can customize the layout of your XML editors. Select **Enable Auto Layout** in the upper right corner and click one of the patterns.

7. Click **OK**.

   The test results shown in Figure 37–49 appear.

   For this example, the source XML and target XML display side-by-side with the XSL map underneath (the default setting). Additional source XML files corresponding to the **Parameters With Schema** table are displayed as tabs in the same area as the main source file. You can right-click an editor and select **Validate XML** to validate the source or target XML against the map source or target XSD schema.

Figure 37–49 Test Results



Note: If the XSL map file contains domain value map (DVM) and cross reference (XREF) XPath functions, it cannot be tested. These functions cannot be executed during design time; they can only be executed during runtime.

37.4.2 How to Generate Reports

You can generate an HTML report with the following information:

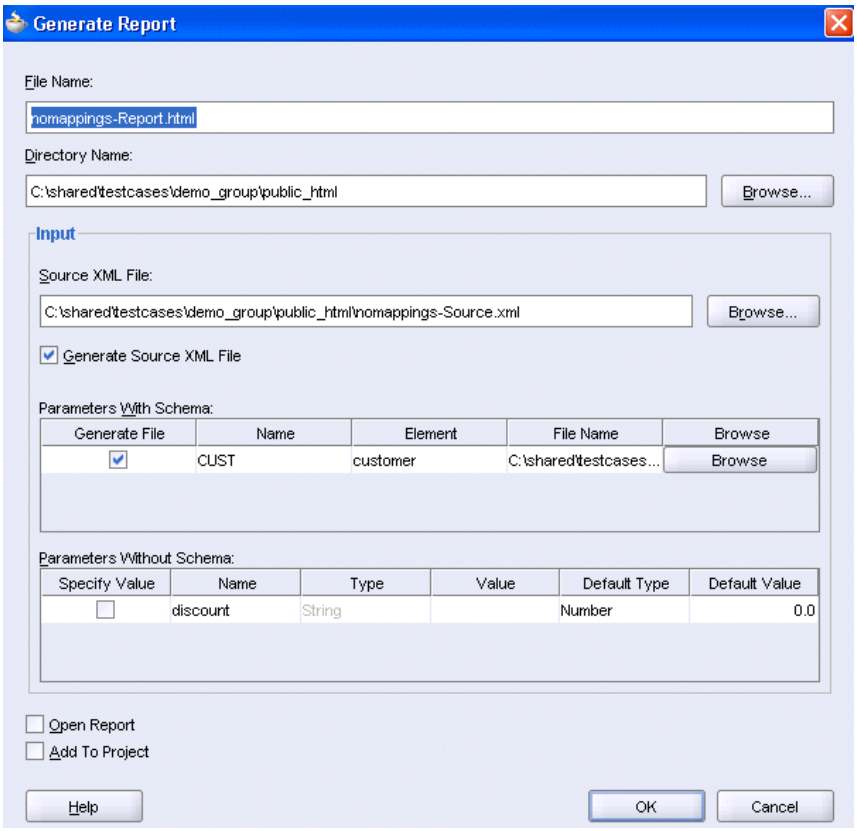
- XSL map file name, source and target schema file names, their root element names, and their root element namespaces
- Target document mappings
- Target fields not mapped (including mandatory fields)
- Sample transformation map execution

Follow these instructions to generate a report.

1. In the center panel, right-click and select **Generate Report**.

   The Generate Report dialog appears, as shown in Figure 37–50. Note that if the map has defined parameters, the appropriate parameter tables appear.

Figure 37–50 The Generate Report Dialog



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

37.4.2.1 Correcting Memory Errors When Generating Reports

If you attempt to generate a report and receive an out-of-memory error, increase the heap size of the JVM as follows.

To increase the JVM heap size:

1. Open the *JDev_Oracle_Home*`\jdev\bin\jdev.conf` file.
2. Go to the following section:

```
# Set the maximum heap to 512M
#
AddVMOption     -Xmx512M
```

3. Increase the size of the heap as follows (for example, to `1024`):

```
AddVMOption     -Xmx1024M
```

In addition, you can also unselect the **Open Report** option on the Generate Report dialog before generating the report.

37.4.3 How to Customize Sample XML Generation

You can customize sample XML generation by specifying the following parameters. Select **Preferences** > **XSL Maps** in the **Tools** main menu of Oracle JDeveloper to display the Preferences dialog.

- Number of repeating elements

  Specifies how many occurrences of an element are created if the element has the attribute `maxOccurs` set to a value greater than `1`. If the specified value is greater than the value of the `maxOccurs` attribute for a particular element, the number of occurrences created for that particular element is the `maxOccurs` value, not the specified number.

- Generate optional elements

  If selected, any optional element (its attribute `minOccurs` set to a value of `0`) is generated the same way as any required element (its attribute `minOccurs` set to a value greater than `0`).

- Maximum depth

  To avoid the occurrence of recursion in sample XML generation caused by optional elements, specify a maximum depth in the XML document hierarchy tree beyond which no optional elements are generated.

37.5 Demonstrating Features of the XSLT Mapper

This sample demonstrates the following features of the XSLT mapper:

- Element and type substitution
- Multiple sources use
- New XSL constructs `xsl:sort` and `xsl:copy-of`
- New variable use

In addition to this sample, Oracle provides other transformation samples that are available for download from the Oracle Technology Network (OTN). These samples are described in Table 37–2. To access these samples, visit the following URL:

`https://soasamples.samplecode.oracle.com/`

Table 37–2 Transformation Samples

Sample	Description
mapper-101-basic-mapping	Demonstrates creation and basic editing of an XSLT map.
mapper-102-import-and-test	Demonstrates the following XSL mapper features: ■ Import of external XSL ■ Test execution with an overview of XML editor validation ■ Report execution
mapper-104-auto-mapping	Demonstrates the automatic mapping feature of the XSLT Mapper.
mapper-105-multiple-sources	Demonstrates the use of multiple sources. The following topics are also covered in the process of creating the map sample. ■ Inserting a cloned `for-each` ■ Adding predicates to XPath expressions ■ Using variables

Table 37–2 (Cont.) Transformation Samples

Sample	Description
mapper-107-extension-functions	Demonstrates the use of user-defined extension functions.
mapper-108-substitution-mapping	Demonstrates the use of element substitution when: ■ An element is defined as the head of a substitution group in the underlying schema. The element may or may not be abstract. Any element from the substitution group can be substituted for the original element. ■ An element is defined as an `any` element. Any global element defined in the schema can be substituted for the `any` element. This is subject to any namespace constraints placed on the definition of the `any` element.
mapper-109-whats-new	Demonstrates new 11*g* features in the XSLT Mapper. These features are described in Section 37.5.1, "Opening the Application" through Section 37.5.7, "Testing the Map."

37.5.1 Opening the Application

You first create the sample application. When complete, the application matches the one provided in the `WhatsNewApplication` directory described in Step 1.

1. Download sample `mapper-109-whats-new` from OTN.

   The sample includes the following files and directories:

   - `artifacts/schemas/po.xsd` and `Attachment.xsd`: source schemas
   - `artifacts/schemas/invoice.xsd` and `ReasonCodes.xsd`: target schemas
   - `artifacts/application`: starting application for this sample
   - `WhatsNewApplication` directory: completed sample map
2. Copy the `artifacts/application` folder to a separate work area.
3. Start Oracle JDeveloper.
4. Click `WhatsNewApplication.jws` in the `artifacts/application` folder you copied to a separate area.
5. If prompted to migrate files, click **Yes**.

   The **WhatsNewApplication** displays in the Application Navigator.

37.5.2 Creating a New XSLT Map in the BPEL Process

You now create a new XSLT map with two sources that is invoked from the BPEL process included in the **WhatsNewApplication** application.

1. In the Application Navigator, double-click the **ProcessPO2Invoice.bpel** BPEL process.
2. From the **Oracle Extensions** section of the Component Palette, drag a **Transform** activity below the **SetDiscontinuedProducts** assign activity.
3. Double-click the **Transform** activity.
4. In the **Name** field of the **General** tab, enter `Po2Invoice`.
5. In the **Transformation** tab, perform the following steps:

a. Click the **Add** icon.

b. From the **Source Variable** list, select **inputVariable**.

c. From the **Source Part** list, select **payload**.

   This variable contains the purchase order that is input to the BPEL process.

d. Click **OK**.

e. Click the **Add** icon a second time and select **DiscontinuedList** from the **Source Variable** list. The variable is created in the **SetDiscontinuedProducts** assign activity before the transformation activity.

f. Click **OK**.

g. From the **Target Variable** list, select **outputVariable**. This is the invoice that is returned from the BPEL process.

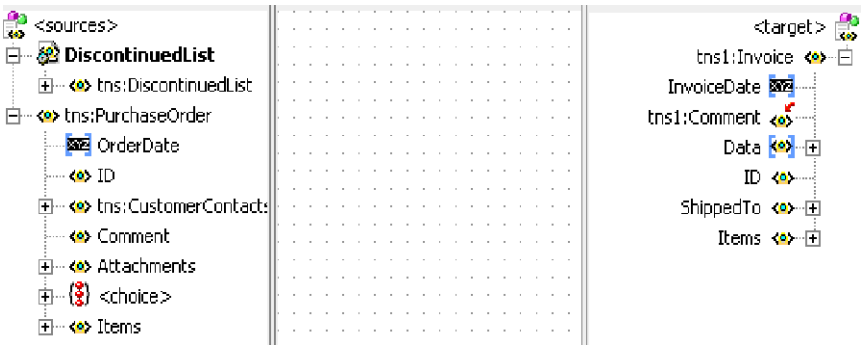
h. In the **Mapper File** field, change the name to `xsl/Po2Invoice`.

i. Click the **Create Mapping** icon to the right of the **Mapper Name** field to create and open the mapper file.

   The XSLT Mapper opens.

j. From the **File** menu, select **Save All**. Your map looks as shown in Figure 37–51. Note that the second source is loaded as a parameter with the name **DiscontinuedList**:

Figure 37–51 XSLT Mapper File



37.5.3 Using Type Substitution to Map the Purchase Order Items

You now use type and element substitutions to map the purchase order items to the invoice items.

1. In the target tree, expand the tree so that **Invoice/Items/Item** is visible. Note that the **Item** element has an error icon next to it.

2. Move the mouse over the element to display a tool tip indicating that this element is defined as an abstract type.

   To map to the **Item** element, you must first indicate which type the element takes in the final XML output.

3. Perform the following steps to indicate which type the element takes:

   a. Right-click the **Item** element and select **Substitute Element or Type**.

      The Substitute Element or Type dialog appears.

b. Select **ShippedItemType** from the list and click **OK**.

The **Item** element structure is filled out. The **xsi:type** attribute sets the type of the **Item** element in the target XML.

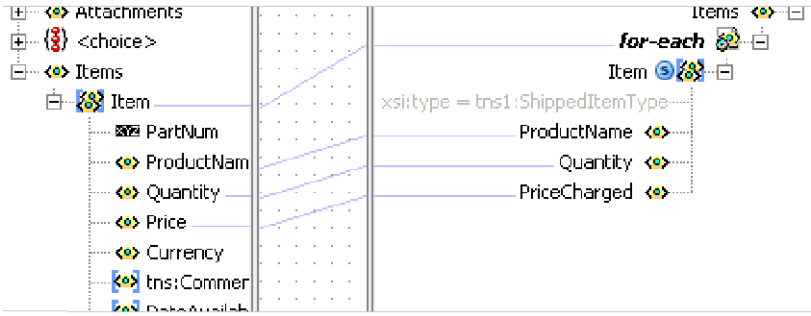
Note: If you view **invoice.xsd**, note that **ShippedItemType** is derived from the abstract type **ItemType**, which is the type of the **Item** element.

---

4. Drag **PurchaseOrder/Items** to **Invoice/Items** to invoke the automatic mapper to map these nodes. To review automatic mapping functionality, see sample mapper-104-auto-mapping.

When complete, the **Item** elements in your map now look as shown in Figure 37–52:

Figure 37–52 Item Elements in XSLT Mapper



5. From the **File** menu, select **Save All** to save the map file.

37.5.4 Referencing Additional Source Elements

You now use the information in the additional source variable, **DiscontinuedList**, to eliminate items that have been discontinued. If the product name for an item is in **DiscontinuedList**, then that item cannot be shipped and is not placed in the final shipped item list.

1. Add an **if** statement above the **Item** node in the target tree by right-clicking the **Item** node and selecting **Add XSL Node** > **if**.

The **if** statement must test if a discontinued product exists in **DiscontinuedList** with the name of the current item. The item is added only to the shipped items if it is not in **DiscontinuedList**. There are many ways to define the test expression for the **if** statement. One way is described in the following steps.

2. Define the test expression for the **if** statement by selecting the following (note that the method for how variables are set has changed from the previous version of Oracle JDeveloper):

a. Add a global variable to the target tree by right-clicking the **Invoice** node and selecting **Add Variable.**

The Add Variable dialog appears.

b. In the **Local Name** field, enter `DelimitedList`. In the following steps, this variable is set to a string with a delimited list of discontinued product names.

c. Click **OK**.

The variable is added with a warning icon next to it.

d. To set the value of the variable, drag the **create-delimited-string** function from the **String** section of the Component Palette to the center panel.

e. Drag **DiscontinuedList/ProductName** to the input side of the **create-delimited-string** function.

f. Drag the output side of the **create-delimited-string** function to the new variable named **DelimitedList**.

g. Double-click the **create-delimited-string** function to open the Edit Function dialog.

h. In the **delimiter** field, add the pipe ("|") character.

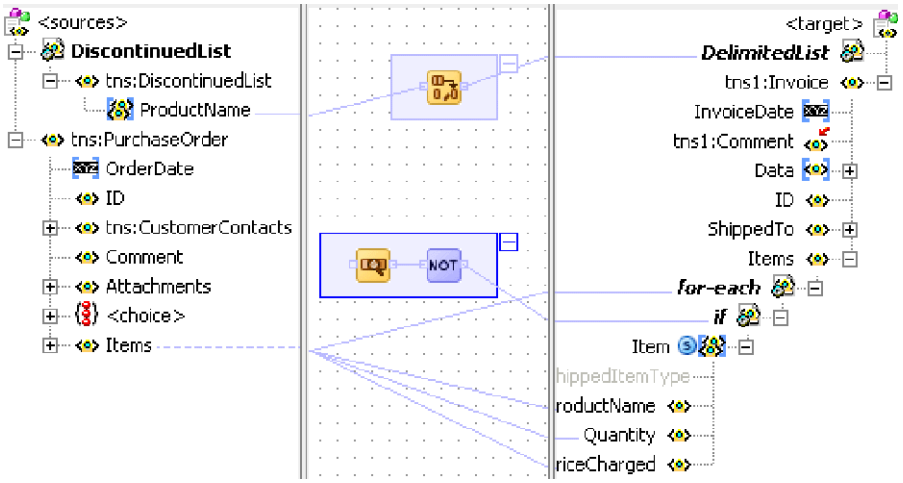
i. Click **OK**.

Note that the input source is referenced in XPath expressions with **$DiscontinuedList**. This source is referenced as an input parameter in XPath expressions.

3. To set the XPath expression for the **if** statement, drag the **contains** function from the **String** section of the Component Palette to the center panel.
4. Drag the **not** function from the **Logical Functions** section of the Component Palette to the shaded area surrounding the **contains** function you added in Step 3.
5. Drag a line from the output side of the **contains** function to the input side of the **not** function.
6. Drag a line from the output side of the **not** function to the **if** statement.
7. Double-click the **contains** function to open the Edit Function dialog.
8. Enter values for the **inputString** and **searchString**, and click **OK**.
9. From the **File** menu, select **Save All** to save the map file.

The map file now looks as shown in Figure 37–53.

Figure 37–53 Mapper File



37.5.5 Using Element Substitution to Map the Shipping Address

You now map a substituted shipping contact element in the source to the **ShippedTo** element in the target.

1. Expand the **PurchaseOrder/CustomerContacts** element in the source to see the **Contact** element.

   Note that this element has an error icon next to it.

2. Place the mouse over this element to display a tool tip indicating that this element is abstract.

   In this situation, you must perform an element substitution to map the element.

3. Right-click the **Contact** element in the source tree and select **Substitute Element or Type**.

   The Substitute Element or Type dialog is displayed with a list of elements in the substitution group of the abstract element **Contact**.

4. Select **ShipToContact** and click **OK**.

   This is the element that you expect in the input XML. The structure of the **ShipToContact** element is now displayed in the source tree.

5. Expand the **ShipToContact/InternationalAddress** element in the source tree to show the address fields.

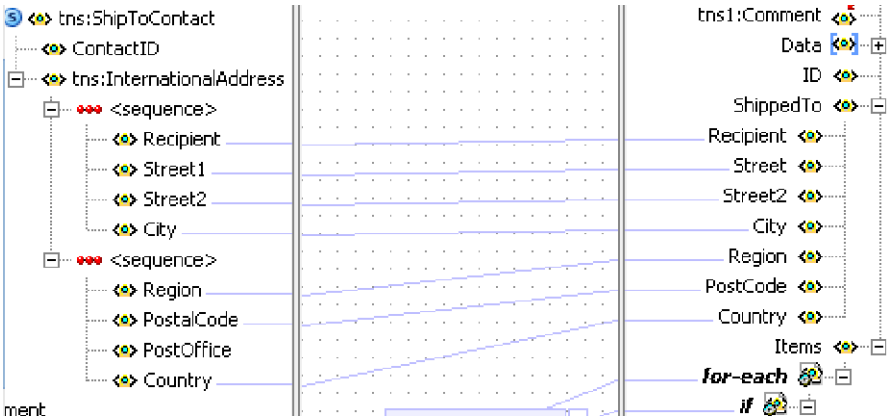
6. Expand the **ShippedTo** element in the target tree to show the target address fields.

   Note the similarity in field names here, indicating that the automatic mapper can be used.

7. Drag the **InternationalAddress** element in the source tree to the **ShippedTo** element in the target tree and use the automatic mapper to help map the address fields below these elements.

8. Map any remaining elements not matched by the automatic mapper so that this section of the map is as shown in Figure 37–54:

Figure 37–54 XSLT Mapper



9. From the **File** menu, select **Save All** to save the map file.

37.5.6 Mapping the Remaining Fields

1. Map **PurchaseOrder/ID** to **Invoice/ID**.

2. Expand **Invoice/Data** to show an **any** element.

3. Use the **copy-of xsl** statement to copy the attachment data from the source to the target **any** element:

a. Right-click the **Invoice/Data/any** element and select **Add XSL Node** > **copy-of**.

   The **copy-of** statement is added and the original **any** element is grayed out. This indicates that it is to be replaced by the nodes selected by the **copy-of** statement.

b. To set the **copy-of** selection, drag the **PurchaseOrder/Attachments** element in the source tree to the **copy-of** statement.

4. Perform the following steps to map the **PurchaseOrder/Comment** field to the **Invoice/Comment** field. Note that the **Invoice/Comment** field is an **anyType** element.

   a. Right-click the **Invoice/Comment** field and select **Substitute Element or Type**.

   b. Select **xsd:string** from the list of types provided.

   c. Drag the **PurchaseOrder/Comment** field to the **Invoice/Comment** field to map the fields.

5. Add an XSL **sort** statement to the **for-each** statement:

   a. Right-click the **for-each** statement in the target tree and select **Add XSL Node** > **sort**.

      The Sort Edit dialog appears.

   b. Select **sort according to data-type Number**.

   c. Select **sort order Descending**.

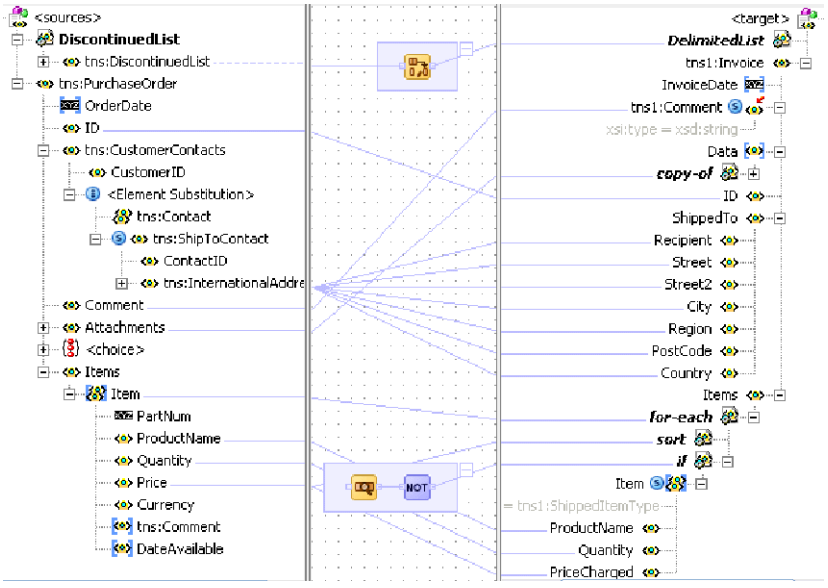
   d. Click **OK**. The sort node is added to the target tree.

   e. Drag **PurchaseOrder/Items/Item/Price** from the source tree to the sort node in the target tree.

      This sets the field on which to sort.

6. From the **File** menu, select **Save All** to save the map file. The map now looks as shown in Figure 37–55:

Figure 37–55 XLST Mapper



37.5.7 Testing the Map

An XSL map can be tested independently from the BPEL process in Oracle JDeveloper using the XSLT Mapper. XML files can be input for each source input to the map.

1. Right-click the center panel and select **Test**.

   The Test XSL Map dialog appears after a warning dialog. The warning indicates that you can test the map by creating your own sample input XML. The sample XML generator cannot generate sample data for the source tree substitutions.

   A sample input XML file is provided: **artifacts/xml/POInput.xml**.

2. Follow these steps to select the sample input file for testing:

   a. Uncheck the **Generate Source XML File** checkbox.

   b. Click the **Browse** button for the **Source XML File** field.

   c. Navigate to select the **artifacts/xml/POInput.xml** file.

   A second sample file has been created with discontinued item data. This file is **artifacts/xml/DiscontinuedItems.xml**.

3. Follow these steps to use this file as the second source input.

   a. Uncheck the **Generate File** checkbox to the left of the **DiscontinuedList** parameter name in the **Parameters With Schema** section of the dialog.

   b. Click **Browse** for the **DiscontinuedList** parameter and select the **artifacts/xml/DiscontinuedItems.xml** file.

4. Click **OK** on the Test XSL Mapper dialog to run the test.

   A **PO2Invoice-Target.xml** file is generated by the execution of the map. Note the use of **xsi:type** attributes, the **Attachments** node created by the **copy-of** statement, and the ordering of items caused by the **sort** statement in the **PO2Invoice-Target.xml** file.

38

Using Business Events and the Event Delivery Network

This chapter describes how to publish and subscribe to business events in a SOA composite application. Business events consist of message data sent as the result of an occurrence in a business environment. When a business event is published, other service components can subscribe to it.

This chapter includes the following sections:

For samples that show how to use business events with Oracle Mediator, visit the following URL:

`https://soasamples.samplecode.oracle.com/`

38.1 Introduction to Business Events

You can raise business events when a situation of interest occurs. For example, in a loan flow scenario, a BPEL process service component executing a loan process can raise a *loan completed event* at the completion of the process. Other systems within the infrastructure of this application can listen for these events and, upon receipt of one instance of an event:

- Use the event context to derive business intelligence or dashboard data.
- Signal to a mail department that a loan package must be sent to a customer.
- Invoke another business process.
- Send information to Oracle Business Activity Monitoring (BAM)

Business events are typically a one-way, fire-and-forget, asynchronous way to send a notification of a business occurrence. The business process does *not*:

- Rely on any service component receiving the business event to complete.
- Care if any other service components receive the business event.

- Need to know where subscribers (if any) are and what they do with the data.

These are important distinctions between business events and direct service invocations that rely on the Web Services Description Language (WSDL) file contract (for example, a SOAP service client). If the author of the event depends on the receiver of the event, then messaging typically must be accomplished through service invocation rather than through a business event. Unlike direct service invocation, the business event separates the client from the server.

A business event is defined using the event definition language (EDL). EDL is a schema used to build business event definitions. Applications work with instances of the business event definition.

EDL consists of the following:

- Global name

  Typically a Java package name (for example, `com.acme.ExpenseReport.created`), though this is not required.

- Payload definition

  The most common use for a definition is an XML Schema (XSD). The payload of a business event is defined using an XSD. The schema URI is contained in the root element of the payload.

Example 38–1 shows an EDL file with two business events in the `BugReport` event definition: `bugUpdated` and `bugCreated`. The namespace (`BugReport`) and associated schema file (`BugReport.xsd`) are referenced.

Example 38–1 EDL File with Two Business Events

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<definitions targetNamespace="/model/events/edl/BugReport"
 xmlns:ns0="/model/events/schema/BugReport"
 xmlns="http://schemas.oracle.com/events/edl">
   <schema-import namespace="/model/events/schema/BugReport"
 location="BugReport.xsd"/>

   <event-definition name="bugCreated">
      <content element="ns0:bugCreatedInfo"/>
   </event-definition>

   <event-definition name="bugUpdated">
      <content element="ns0:bugUpdatedInfo"/>
   </event-definition>
</definitions>
```

These two events are available for subscription in Oracle Mediator.

Business events are deployed to the metadata service (MDS) repository. Deploying a business event to MDS along with its artifacts (for example, the XSDs) is known as publishing the EDL (or event definition). This action transfers the EDL and its artifacts to a shared area in MDS. An object in an MDS shared area is visible to all applications in the Resource Palette of Oracle JDeveloper. After an EDL is published, it can be subscribed to by other applications. EDLs cannot be unpublished; the definition always exists.

A subscription is for a specific qualified name (QName) (for example, `x.y.z/newOrders`). A QName is a tuple (`URI`, `localName`) that may be derived from a string `prefix:localName` with a namespace declaration such as

`xmlns:prefix=URI` or a namespace context. In addition, subscriptions can be further narrowed down by using content-based filters.

Business events are published in the Event Delivery Network (EDN). The EDN runs within every SOA instance. Raised events are delivered by EDN to the subscribing service components. Oracle Mediator service components and BPEL process service components can subscribe to and publish events.

The EDN has two different implementations:

- EDN-DB: Uses an Oracle database as a back-end store and depends on Oracle-specific features.
- EDN-JMS: Uses a generic JMS queue as a back-end store.

If you are using an Oracle database, Oracle recommends that you use EDN-DB instead of EDN-JMS.

38.1.1 Local and Remote Events Boundaries

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

Raising an event from a Java EE application can be done through a local event connection or a remote event connection:

- Local event connection

  If the publisher resides on the same Oracle WebLogic Server as the application and the publisher uses a local business event connection factory, the event is raised through a local event connection. In this scenario, synchronous subscriptions are executed synchronously.

- Remote event connection

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

You can also raise events through PL/SQL APIs.

If another application (for example, an Oracle ADF Business Component application) is configured to run in the same container as the SOA composite application, it is optimized to use local event connections. The boundary for events is the application instance. When an event is raised in the application instance, subscriptions registered in the application instance are executed. Events are not propagated from one application instance to another. Propagation can be achieved through an Oracle Mediator in both instances, which listens for events and publishes them to a JMS queue.

38.2 Creating Business Events in Oracle JDeveloper

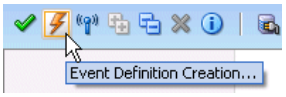
This section provides a high-level overview of how to create and subscribe to a business event. In this scenario, a business event named NewOrderSubmitted is created when a user places an order in a store front application (StoreFrontService service). You then create an Oracle Mediator service component to subscribe to this business event.

38.2.1 How to Create a Business Event

To create a business event:

1. Create a SOA project as an empty composite.
2. Launch the Event Definition Creation wizard in either of two ways:
   a. In the SOA Composite Editor, click the icon above the designer. Figure 38–1 provides an example.

Figure 38–1 Event Definition Creation



   b. From the **File** main menu, select **New** > **SOA Tier** > **Service Components** > **Event Definition**.

   The Event Definition Creation dialog appears.
3. Enter the details described in Table 38–1.

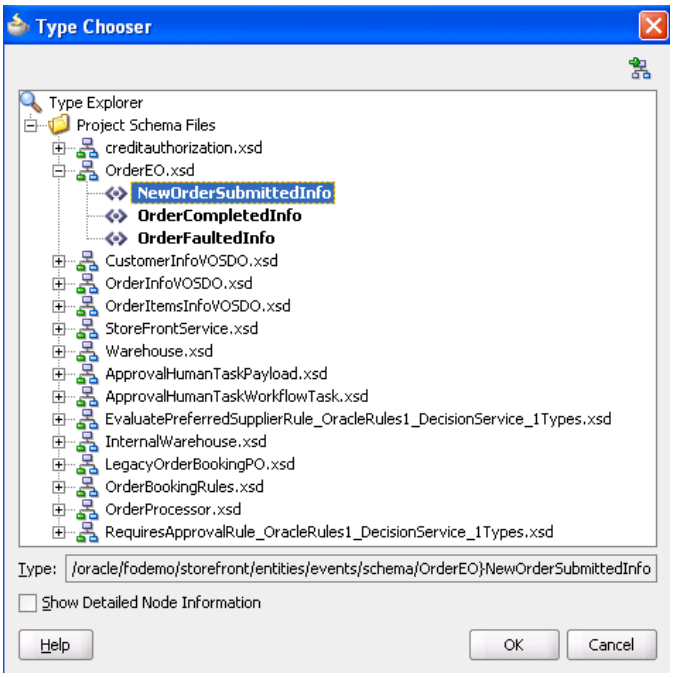
Table 38–1 Event Definition Creation Wizard Fields and Values

Field	Value
Event Definition Name	Enter a name. **Note:** Do not enter a forward slash (`/`) as the event name. This creates an event definition file consisting of only an extension for a name (`.edn`).
Directory	Displays the directory path.
Namespace	Accept the default namespace or enter a value for the namespace in which to place the event.

4. Click the **Add** icon to add an event.

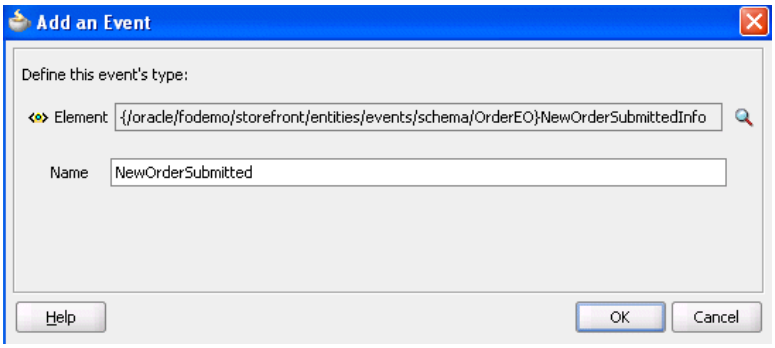
   The Add an Event dialog appears.
5. Click the **Search** icon to select the payload, and click **OK**. Figure 38–2 provides details.

Figure 38–2 Select the Payload



6. In the **Name** field, enter a name, as shown in Figure 38–3.

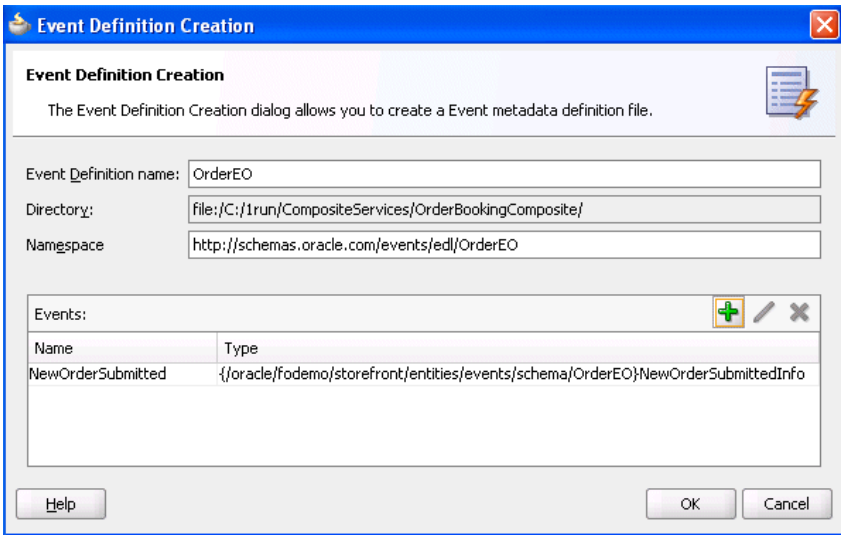
Figure 38–3 Add an Event Dialog



7. Click **OK**.

   The added event now appears in the **Events** section, as shown in Figure 38–4.

Figure 38–4 Event Definition Creation Dialog



8. Above the editor, click the cross icon (**x**) next to *event_definition_name*.edl to close the editor.
9. Click **Yes** when prompted to save your changes. If you do not save your changes, the event is not created and cannot be selected in the Event Chooser window.

   The business event is published to MDS and you are returned to the SOA Composite Editor. The business event displays for browsing in the Resource Palette.

38.3 Subscribing to or Publishing a Business Event from an Oracle Mediator Service Component

This section describes how to subscribe to a business event or publish a business event from an Oracle Mediator service component.

38.3.1 How to Subscribe to a Business Event

To subscribe to a business event:

1. From the Component Palette, drag a **Mediator** service component into the SOA Composite Editor. This service component enables you to subscribe to the business event.
2. In the **Name** field, enter a name.
3. From the **Options** list, select **Subscribe to Event**.

   The window is refreshed to display an events table.
4. Click the **Add** icon to select an event.

   The Event Chooser window appears.
5. Select the event you created and click **OK**.

   You are returned to the Create Mediator dialog.
6. Select a level of delivery consistency for the event.
   - **one and only one**

Events are delivered to the subscriber in its own global (that is, JTA) transaction. Any changes made by the subscriber within that transaction are committed after the event processing is complete. If the subscriber fails, the transaction is rolled back. Failed events are retried a configured number of times.

- **guaranteed**

  Events are delivered to the subscriber asynchronously without a global transaction. The subscriber can choose to create its own local transaction for processing, but it is committed independently of the rest of the event processing. The event is guaranteed to be handed to the subscriber, but because there is no global transaction, there is a possibility that a system failure can cause an event to be delivered more than once. If the subscriber throws an exception (or fails in any way), the exception is logged, but the event is not resent.

- **immediate**

  Events are delivered to the subscriber in the same global transaction and same thread as the publisher. The publish call does not return until all immediate subscribers have completed processing. If any subscribers throw an exception, no additional subscribers are invoked and an exception is thrown to the publisher. The transaction is rolled back in case of any error during immediate processing.

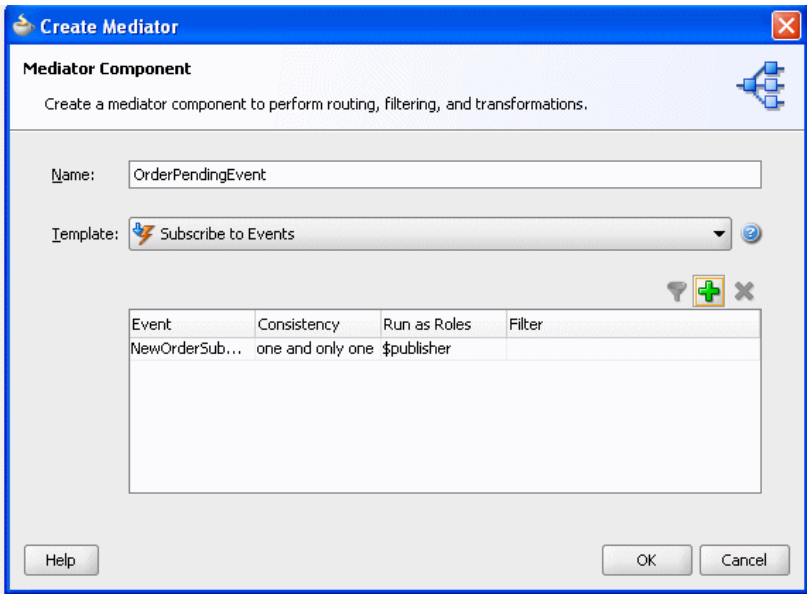
7. If you want to filter the event, double-click the **Filter** column of the selected event or select the event and click the **filter** icon (first icon) above the table. This displays the Expression Builder dialog. This dialog enables you to specify an XPath filter expression. A filter expression specifies that the contents (payload or headers) of a message be analyzed before any service is invoked. For example, you can apply a filter expression that specifies that a service be invoked only if the message includes a customer ID.

   When the expression logic is satisfied, the event is accepted for delivery.

   For more information about filters, see Section 19.2.2.7, "How to Specify an Expression for Filtering Messages."

   Figure 38–5 shows the Create Mediator dialog.

Figure 38–5 Create Mediator Dialog



8. Click **OK**.

   Figure 38–6 shows an icon on the left side that indicates that Oracle Mediator is configured for an event subscription.

Figure 38–6 Configuration for Event Subscription

38.3.2 What Happens When You Create and Subscribe to a Business Event

The source code in Example 38–2 provides details about the subscribed event of the Oracle Mediator service component.

Example 38–2 Subscribed Event

```
<component name="OrderPendingEvent">
    <implementation.mediator src="OrderPendingEvent.mplan"/>
    <business-events>
      <subscribe
         xmlns:sub1="/oracle/fodemo/storefront/entities/events/edl/OrderEO"
         name="sub1:NewOrderSubmitted" consistency="oneAndOnlyOne"
         runAsRoles="$publisher"/>
</business-events>
</component>
```

While not explicitly demonstrated in this example, you can define XPath filters on events. In Example 38–3, the event is accepted for delivery *only* if the initial deposit is greater than `50000`:

Example 38–3 Definition of XPath Filters on Events

```
<business-events>
    . . .
    . . .
    <filter>
        <xpath xmlns:be="http://oracle.com/fabric/businessEvent"
             xmlns:ns1="http://xmlns.oracle.com/singleString"
               <xpath expression= "/be:business-event/be:content/
               sub1:AccountInfo/Details[@initialDeposit > 50000]" />
    </filter>
  . . .
  . . .
</business-events>
```

38.3.3 What You May Need to Know About Subscribing to a Business Event

Subscribers in nondefault revisions of SOA composite applications can still get business events. For example, note the following behavior:

1. Create a composite application with an initial Oracle Mediator service component named M1 that publishes an event and a second Oracle Mediator service component named M2 that subscribes to the event. The output is written to a directory.
2. Deploy the composite application as revision 1.
3. Modify the composite application by adding a third Oracle Mediator service component named M3 that subscribes to the same event and writes the output to a different directory.
4. Deploy the composite application as revision 2 (the default).
5. Invoke revision 2 of the composite application.

   Note that Oracle Mediator M2 writes the output to two files with the same content in the directory. As expected, Oracle Mediator M3 picks up the event and writes the output successfully to another directory. However, note that Oracle Mediator M2 (from revision 1) is also picking up and processing the published event from revision 2 of the composite application. Therefore, it creates one more output file in the same directory.

38.3.4 How to Publish a Business Event

You can create a second Oracle Mediator to publish the event that you subscribed to in Section 38.3.1, "How to Subscribe to a Business Event."

To publish a business event:

1. Create a second Oracle Mediator service component that publishes the event to which the first Oracle Mediator subscribes.
2. Return to the first Oracle Mediator service component.
3. In the **Routing Rules** section, click the **Add** icon.
4. Click **Service** when prompted by the Target Type window.
5. Select the second Oracle Mediator service component.
6. From the **File** main menu, select **Save All**.

38.3.5 How to Configure a Foreign JNDI Provider to Enable Administration Server Applications to Publish Events to the SOA Server

This section describes how to configure a foreign JNDI Provider when the publishing application (for example, an ADF EAR file) is deployed on the administration server instead of the SOA server.

To configure a foreign JNDI provider to enable administration server applications to publish events to the SOA Server:

1. Log in to the Oracle WebLogic Server Administration Console.

   `http://host:port/console`

2. In the **Domain Structure** section, expand **Services** > **Foreign JNDI Providers**.
3. Click **Lock & Edit**.
4. Click **New**.
5. In the **Name** field, enter a name (for example, `SOA_JNDI`), and click **Next**.
6. Select the **AdminServer** checkbox, and click **Finish**.
7. In the **Name** column, click the provider name you entered in Step 5.
8. Enter the details shown in Table 38–2, and click **Save**.

Table 38–2 Configuration Details

Field	Description
Initial Context Factory	Enter `weblogic.jndi.WLInitialContextFactory`.
Provider URL	Enter `t3://hostname:soa_server_port`.
User	Enter the Oracle WebLogic Server user name.
Password and **Confirm Password**	Enter the password for the Oracle WebLogic Server user name.

9. Click **Links** > **New**.
10. Enter the details shown in Table 38–3, and click **OK**.

Table 38–3 Configuration Details

Field	Description
Name	Enter `SOA_EDNDataSource`.
Local Name	Enter `jdbc/EDNDataSource`.
Remote Name	Enter `jdbc/EDNDataSource`.

11. Click **New**.
12. Enter the details shown in Table 38–4, and click **OK**.

Table 38–4 Configuration Details

Field	Description
Name	Enter `SOA_EDNLocalTxDataSource`.
Local Name	Enter `jdbc/EDNLocalTxDataSource`.

Table 38–4 (Cont.) Configuration Details

Field	Description
Remote Name	Enter `jdbc/EDNLocalTxDataSource`.

13. Click **OK**.
14. Click **Activate Changes**.
15. Modify the *FMW_Home*`/user_projects/domains/`*domain_name*`/bin/setDomainEnv.sh` file for Linux (or `setDomainEnv.bat` file for Windows) as follows:

    ```
    WLS_JDBC_REMOTE_ENABLED="-Dweblogic.jdbc.remoteEnabled=true"
    ```

16. Restart the server.

38.3.6 How to Configure JMS-based EDN Implementations

The following JNDI configuration changes are required when the EDN implementation is JMS-based (EDN-JMS). In these scenarios, a generic JMS queue is used as the back-end store. These changes enable the remote client (for example, the ADF application client) to look up the connection factory before publishing events.

To configure JMS-based EDN Implementations

1. Log in to the Oracle WebLogic Server Administration Console.

   `http://`*host*`:`*port*`/console`

2. In the **Domain Structure** section, expand **Services** > **Data Sources**.

   You must remove the EDN-DB JNDI sources to use EDN-JMS data sources.

3. Select the following EDN-DB JNDI data sources, and click **Remove**.

   - `jdbc/EDNDataSource`
   - `jdbc/EDNLocalTxDataSource`

   If the event publisher is in an application (for example, ADF) running in a different cluster or even in a different domain from the SOA server for EDN, you must configure a foreign JNDI provider with the local JNDI names for the cluster mapping to JNDI names targeted to the SOA Infrastructure. Local and remote JNDI names are the same in the links.

4. In the **Domain Structure** section, expand **Services** > **Foreign JNDI Providers**.
5. Click **New**.
6. In the **Name** field, enter a name for the foreign JNDI provider.
7. Select targets for the new JNDI provider, and click **Finish**.
8. In the **Name** field, click the new JNDI provider.
9. Specify provider settings (the initial context factory, provider URL, and so on), and click **Save**.
10. Click the **Links** tab.
11. Click **New** to create a foreign JNDI link.
12. Enter a name, then specify the local and remote JNDI name of `jms/fabric/EDNConnectionFactory`.

13. Repeat Step 12, and specify a name and the local and remote JNDI name of `jms/fabric/xaEDNConnectionFactory`.
14. Repeat Step 12, and specify a name and the local and remote JNDI name of `jms/fabric/EDNQueue`.

    Once complete, three links are created.
15. Restart the targeted servers.
16. Confirm the new JNDI provider links in the JNDI tree view.

If you do not make these configuration changes, errors similar to those shown in Example 38–4 occur.

Example 38–4 EDN-JMS Error Messages

```
<Aug 30, 2010 1:28:46 PM EDT> <Warning>
<oracle.adf.controller.faces.lifecycle.Utils> <BEA-000000> <ADF: Adding the
following JSF error message: [FMWGEN][SQLServer JDBC Driver][SQLServer]Could
not find stored procedure 'edn_internal_publish_event'.

oracle.fabric.common.FabricException: Error enqueueing event:
[FMWGEN][SQLServer JDBC Driver][SQLServer]Could not find stored procedure
'edn_internal_publish_event'.

              at
oracle.integration.platform.blocks.event.saq.SAQRemoteBusinessEventConnection.
enqueueEvent(SAQRemoteBusinessEventConnection.java:86)
```

38.3.7 What Happens When You Publish a Business Event

Note that the two Oracle Mediator service components appear in Example 38–5. One service component (`OrderPendingEvent`) subscribes to the event and the other service component (`PublishOrderPendingEvent`) publishes the event.

Example 38–5 Event Subscription and Publication

```
<component name="PublishOrderPendingEvent">
    <implementation.mediator src="PublishOrderPendingEvent.mplan"/>
    <business-events>
      <publishes
xmlns:sub1="/oracle/fodemo/storefront/entities/events/edl/OrderEO"
 name="pub1:NewOrderSubmitted"/>
    </business-events>
  </component>

<component name="OrderPendingEvent">
    <implementation.mediator src="OrderPendingEvent.mplan"/>
    <business-events>
      <subscribe
         xmlns:sub1="/oracle/fodemo/storefront/entities/events/edl/OrderEO"
         name="sub1:NewOrderSubmitted" consistency="oneAndOnlyOne"
         runAsRoles="$publisher"/>
</business-events>
</component>
```

38.4 Subscribing to or Publishing a Business Event from a BPEL Process Service Component

This section describes how to subscribe to a business event or publish a business event from a BPEL process service component.

38.4.1 How to Subscribe to a Business Event

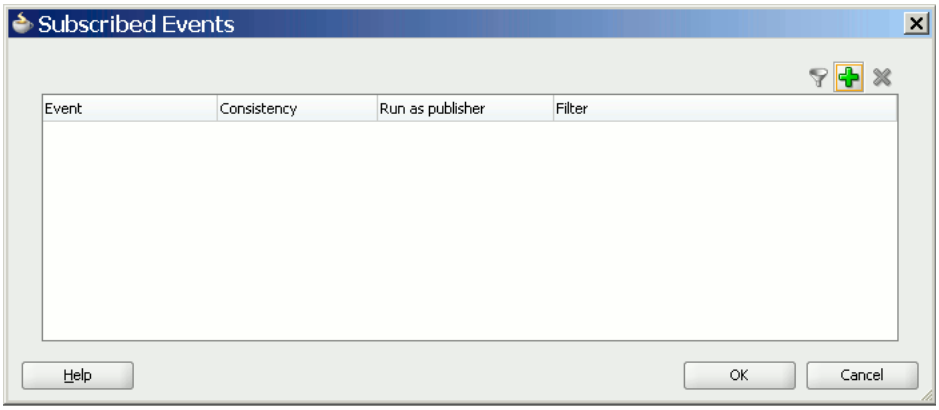
To subscribe to a business event:

1. From the Component Palette, drag a **BPEL Process** service component into the SOA Composite Editor.
2. In the **Name** field, enter a name. Do not change any other default option and click **OK**.

   The BPEL process service component is created.
3. Double-click the BPEL process service component. The Oracle BPEL Designer is opened. Alternatively, you can also right-click the BPEL process service component and click **Edit**.
4. Drag a **Receive** activity from the Component Palette into the SOA Composite Editor, below the **receiveInput** activity.

   Note: The onMessage branch of a pick activity can also be set up to receive events from the EDN. For more information about the onMessage branch, see Section 14.2, "Creating a Pick Activity to Select Between Continuing a Process or Waiting."
5. Double-click the **Receive** activity. The Receive dialog opens. Alternatively, you can also right-click the **Receive** activity and click **Edit**.
6. In the **Name** field, enter a name.
7. From the **Interaction Type** list, select **Event**. The layout of the Receive dialog changes.
8. Click the **Browse Events** icon to the right of the **Event** field. The Subscribed Events dialog appears, as shown in Figure 38–7.

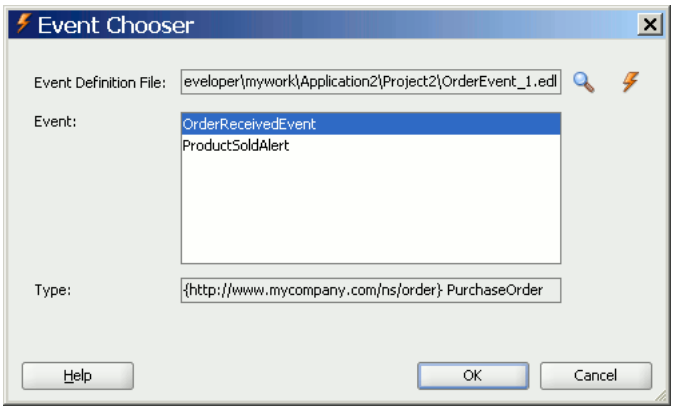
Figure 38–7 Subscribed Events Dialog



9. Click the **Add** icon to select an event.

The Event Chooser dialog appears, as shown in Figure 38–8.

Figure 38–8 Event Chooser Dialog



10. Select the event you created and click **OK**.

    You are returned to the Subscribed Events dialog.

11. Select a level of delivery consistency for the event.

    - **one and only one**

      Events are delivered to the subscriber in its own global (that is, JTA) transaction. Any changes made by the subscriber within that transaction are committed after the event processing is complete. If the subscriber fails, the transaction is rolled back. Failed events are retried a configured number of times.

    - **guaranteed**

      Events are delivered to the subscriber asynchronously without a global transaction. The subscriber can choose to create its own local transaction for processing, but it is committed independently of the rest of the event processing. The event is guaranteed to be handed to the subscriber, but because there is no global transaction, there is a possibility that a system failure can cause an event to be delivered more than once. If the subscriber throws an exception (or fails in any way), the exception is logged, but the event is not resent.

    - **immediate**

      Events are delivered to the subscriber in the same global transaction and same thread as the publisher. The publish call does not return until all immediate subscribers have completed processing. If any subscribers throw an exception, no additional subscribers are invoked and an exception is thrown to the publisher. The transaction is rolled back in case of any error during immediate processing.

12. If you want to filter the event, double-click the **Filter** column of the selected event or select the event and click the **filter** icon (first icon) above the table. This displays the Expression Builder dialog. This dialog enables you to specify an XPath filter expression. A filter expression specifies that the contents (payload or headers) of a message be analyzed before any service is invoked. For example, you can apply a filter expression that specifies that a service be invoked only if the order includes an order ID.

    When the expression logic is satisfied, the event is accepted for delivery.

13. Click **OK** to close the Subscribed Events dialog. You are returned to the Receive dialog.

> **Note:** Optionally, you can select the **Create Instance** checkbox, if this receive activity is the initiating receive activity that starts the BPEL process service component instance. This action enables creation of a new BPEL process service component instance for every invocation.
>
> If this receive activity is a midprocess receive activity in which the BPEL instance is already started, then this receive activity waits for another event to continue the execution of this BPEL instance.

14. Click **OK**.

    Figure 38–9 shows a BPEL process service component that is configured for event subscription.

Figure 38–9 BPEL Process Service component Configuration for Event Subscription

38.4.2 How to Publish a Business Event

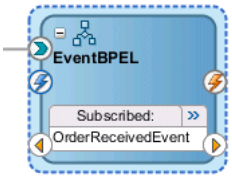
To publish a business event:

1. Drag an **Invoke** activity from the Component Palette into the SOA Composite Editor, below the **Receive** activity created in Section 38.4.1, "How to Subscribe to a Business Event."
2. Double-click the **Invoke** activity. The Invoke dialog opens. Alternatively, you can also right-click the **Invoke** activity and click **Edit**.
3. In the **Name** field, enter a name.
4. From **Interaction Type** list, select **Event**. The layout of the Invoke dialog changes.
5. Click the **Browse Events** icon to the right of the **Event** field. The Event Chooser window appears.
6. Select the event you created and click **OK**.

   You are returned to the Invoke dialog.
7. Click **OK**.

   Figure 38–10 shows a BPEL process service component that is configured for an event subscription and publication. The blue lightning bolt in the circle on the left side indicates event subscription. The yellow lightning bolt in the circle on the right side indicates event publication. Clicking the blue arrow inside the title changes it to display the title of the published event.

Figure 38–10 BPEL Process Service Component Configuration for Event Subscription and Publishing



38.4.3 What Happens When You Subscribe to and Publish a Business Event

The source code in Example 38–6 shows how the `composite.xml` source changes for the subscribed and published events of a BPEL process service component.

Example 38–6 Event Subscription and Publication

```
<component name="EventBPELProcess">
   <implementation.bpel src="EventBPELProcess.bpel"/>
   <property name="configuration.monitorLocation" type="xs:string"
             many="false">EventBPELProcess.monitor</property>
   <business-events>
     <subscribe xmlns:sub1="http://mycompany.com/events/orders"
                name="sub1:OrderReceivedEvent" consistency="oneAndOnlyOne"
                runAsRoles="$publisher"/>
     <publishes xmlns:pub1="http://mycompany.com/events/orders"
                name="pub1:ProductSoldAlert"/>
   </business-events>
</component>
```

While not explicitly demonstrated in this example, you can define XPath filters on events. A filter is typically present in event subscriptions (the `subscribe` element limits the type of event to which this service component is subscribed, and the `filter` section further limits the event to specific content in which the component is interested). In Example 38–7, the event is accepted for delivery *only* if the initial deposit is greater than `50000`.

Example 38–7 Definition of XPath Filters on Events

```
    <business-events>
        . . .
        . . .
        <filter>
            <xpath xmlns:be="http://oracle.com/fabric/businessEvent"
                 xmlns:ns1="http://xmlns.oracle.com/singleString"
                   <xpath expression= "/be:business-event/be:content/
                   sub1:AccountInfo/Details[@initialDeposit > 50000]" />
        </filter>
      . . .
      . . .
    </business-events>
```

The standard BPEL activities such as receive, invoke, onMessage, and onEvent (in BPEL 2.0) are extended with an extra attribute `bpelx:eventName`, so that the BPEL process service component can receive events from the EDN event bus. The schema for the `eventName` attribute is shown in Example 38–8:

Example 38–8 The Schema for the Eventname Attribute

```
<xs:attribute name="eventName" type="xs:QName">
        <xs:annotation>
            <xs:appinfo>
                <tns:parent>
                    <bpel11:onMessage/>
                    <bpel11:receive/>
                    <bpel11:invoke/>
                </tns:parent>
            </xs:appinfo>
        </xs:annotation>
    </xs:attribute>
```

Example 38–9 shows how the `eventName` attribute is used in the BPEL source file:

Example 38–9 BPEL Source Code Using eventName Attribute

```
<receive name="OrderPendingEvent" createInstance="yes"
         bpelx:eventName="ns1:OrderReceivedEvent"/>
<invoke name="Invoke_1" bpelx:eventName="ns1:ProductSoldAlert"/>
```

If the `bpelx:eventName` attribute is used in a receive, invoke, onMessage, or onEvent (in BPEL 2.0) element, then the standard attributes such as `partnerLink`, `operation`, or `portType` attributes are not present. This is because the existence of the `bpelx:eventName` attribute shows that the activity is only interested in receiving events from the EDN event bus or publishing events to the EDN event bus.

The XSD file for the BPEL process service component is slightly modified, so that the `partnerLink`, `operation`, and `portTyp` attributes are no longer mandatory. The Oracle JDeveloper validation logic should enforce the presence of either the `bpelx:eventName` attribute or the `partnerLink`, `operation`, and `portTyp` attributes, but not both. Example 38–10 shows the modified schema definition of a BPEL receive activity.

Example 38–10 Schema Definition of a BPEL Receive Activity

```
<complexType name="tReceive">
        <complexContent>
           <extension base="bpws:tExtensibleElements">
                <sequence>
                    <element name="correlations" type="bpws:tCorrelations"
minOccurs="0"/>
                    <group ref="bpws:activity"/>
                </sequence>
                <!- BPEL mandatory attributes relaxed to optional for supporting
BPEL-EDN ->
                <attribute name="partnerLink" type="NCName" use="optional"/>
                <attribute name="portType" type="QName" use="optional"/>
                <attribute name="operation" type="NCName" use="optional"/>
                <attribute name="variable" type="NCName" use="optional"/>
            </extension>
        </complexContent>
    </complexType>
```

The schema definition for the invoke and onMessage activities are modified similarly.

38.4.4 What You May Need to Know About Subscribing to a Business Event

Note that subscribers in nondefault revisions of SOA composite applications can still get business events. For example, note the following behavior:

1. Create a composite application with an initial Mediator service component or BPEL process service component named S1 that publishes an event and a second Mediator service component or BPEL process service component named S2 that subscribes to the event. The output is written to a directory.
2. Deploy the composite application as revision 1.
3. Modify the composite application by adding a third Mediator service component or BPEL process service component named s3 that subscribes to the same event and writes the output to a different directory.
4. Deploy the composite application as revision 2 (the default).
5. Invoke revision 2 of the composite application.

Note that service component S2 writes the output to two files with the same content in the directory. As expected, service component S3 picks up the event and writes the output successfully to another directory. However, note that service component S2 (from revision 1) also picks up and processes the published event from revision 2 of the composite application. Therefore, it creates one more output file in the same directory.

38.5 How to Integrate Oracle ADF Business Component Business Events with Oracle Mediator

This section provides a high-level overview of how to integrate Oracle ADF Business Component event conditions with SOA components. The SOA components include Mediator service components and BPEL process service components.

To integrate Oracle ADF Business Component business events with SOA Components:

1. Create a business component project.
2. Add a business event definition to the project. This action generates an EDL file and an XSD file. The XSD file contains the definition of the payload. Ensure also that you specify that the event be raised by the Oracle ADF Business Component upon creation.

   For more information about creating and publishing Oracle ADF Business Component business events, see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
3. Create a SOA composite application and manually copy the EDL and XSD schema files to the root directory of the SOA project. For example:

   `JDeveloper/mywork/`*`SOA_application_name`*`/`*`SOA_project_name`*
4. Place schema files at the proper relative location from the EDL file based on the import.
5. Create a Mediator service component as described in Section 38.3.1, "How to Subscribe to a Business Event".
6. In the Event Chooser window, select the EDL file of the event, as described in Section 38.3.1, "How to Subscribe to a Business Event."

7. Create a BPEL process service component in the same SOA composite application for the Oracle Mediator to invoke. In the **Input Element** field of the **Advanced** tab, ensure that you select the payload of the Business Component business event XSD created in Step 2.
8. Double-click the BPEL process service component.
9. Drag an **Email** activity into the BPEL process service component.
10. Use the payload of the business event XSD to complete the **Subject** and **Body** fields.
11. Return to the Oracle Mediator service component in the SOA Composite Editor.
12. Design a second service component to publish the event, such as a BPEL process service component or a second Oracle Mediator service component.

    SOA composite application design is now complete.

Part VIII

Completing Your Application

This part describes how to complete design of your application.

This part contains the following chapters:

39

Enabling Security with Policies

This chapter describes how to manage policies during design-time in SOA composite applications.

This chapter includes the following sections:

39.1 Introduction to Policies

Oracle Fusion Middleware uses a policy-based model to manage and secure Web services across an organization. Policies apply security to the delivery of messages. Policies can be managed by both developers in a design-time environment and system administrators in a runtime environment.

Policies are comprised of one or more assertions. A policy assertion is the smallest unit of a policy that performs a specific action. Policy assertions are executed on the request message and the response message, and the same set of assertions is executed on both types of messages. The assertions are executed in the order in which they appear in the policy.

Table 39–1 describes the supported policy categories.

Table 39–1 Supported Policy Categories

Category	Description
Message Transmission Optimization Mechanism (MTOM)	Ensures that attachments are in MTOM format. This format enables binary data to be sent to and from web services. This reduces the transmission size on the wire.
Reliability	Supports the WS-Reliable Messaging protocol. This guarantees the end-to-end delivery of messages.
Addressing	Verifies that simple object access protocol (SOAP) messages include WS-Addressing headers in conformance with the WS-Addressing specification. Transport-level data is included in the XML message rather than relying on the network-level transport to convey this information.
Security	Implements the WS-Security 1.0 and 1.1 standards. They enforce authentication and authorization of users. identity propagation, and message protection (message integrity and message confidentiality).
Management	Logs request, response, and fault messages to a message log. Management policies can also include custom policies.

Within each category there are one or more policy types that you can attach. For example, if you select the reliability category, the following types are available for selection:

- oracle/wsrm10_policy

  Supports version 1.0 of the Web Services Reliable Messaging protocol

- oracle/wsrm11_policy

  Supports version 1.1 of the Web Services Reliable Messaging protocol

- oracle/no_wsrm_policy

  Supports the disabling of a globally attached Web Services Reliable Messaging policy

For more information about available policies, details about which ones to use in your environment, and global policies, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

39.2 Attaching Policies to Binding Components and Service Components

You can attach or detach policies to and from service binding components, service components, and reference binding components in a SOA composite application. Use Oracle JDeveloper to attach policies for testing security in a design-time environment. When your application is ready for deployment to a production environment, you can attach or detach runtime policies in Oracle Enterprise Manager Fusion Middleware Control.

For more information about runtime management of policies, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

39.2.1 How to Attach Policies to Binding Components and Service Components

To attach a policy to a service or reference binding component:

1. In the SOA Composite Editor, right-click a service binding component or reference binding component.
2. Select **Configure WS-Policies**.

   Depending upon the interface definition of your SOA composite application, you may be prompted with an additional menu of options.

   - If the selected service or reference is interfacing with a synchronous BPEL process or Oracle Mediator service component, a single policy is used for both request and response messages. The Configure SOA WS Policies dialog immediately appears. Go to Step 4.
   - If the service or reference is interfacing with an asynchronous BPEL process or Oracle Mediator service component, the policies must be configured separately for request and response messages. The policy at the callback is used for the response sent from service to client. An additional menu is displayed. Go to Step 3.
3. Select the type of binding to use:
   - **For Request**:

Select the request binding for the service component with which to bind. You can only select a single request binding. This action enables communication between the binding component and the service component.

When request binding is configured for a service in the **Exposed Services** swimlane, the service acts as the server. When request binding is configured for a reference in the **External References** swimlane, the reference acts as the client.

- **For Callback**: (only for interactions with asynchronous processes)

  Select the callback binding for the service component with which to bind. This action enables message communication between the binding component and the service component. You can only select a single callback binding.

  When callback binding is configured for a service in the **Exposed Services** swimlane, the service acts as the client. When callback binding is configured for a reference in the **External References** swimlane, the reference acts as the server.

The Configure SOA WS Policies dialog shown in Figure 39–1 appears. For this example, the **For Request** option was selected for a service binding component. The same types of policy categories are also available if you select **For Callback**.

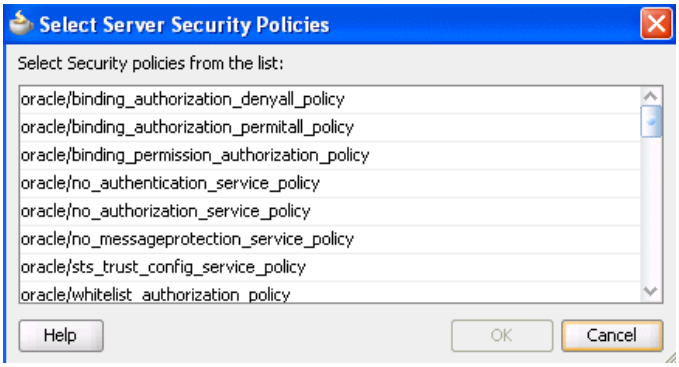
Figure 39–1 Configure SOA WS Policies Dialog

4. Click the **Add** icon for the type of policy to attach:
   - **MTOM**
   - **Reliability**

- **Addressing**
- **Security**
- **Management**

For this example, **Security** is selected. The dialog shown in Figure 39–2 is displayed.

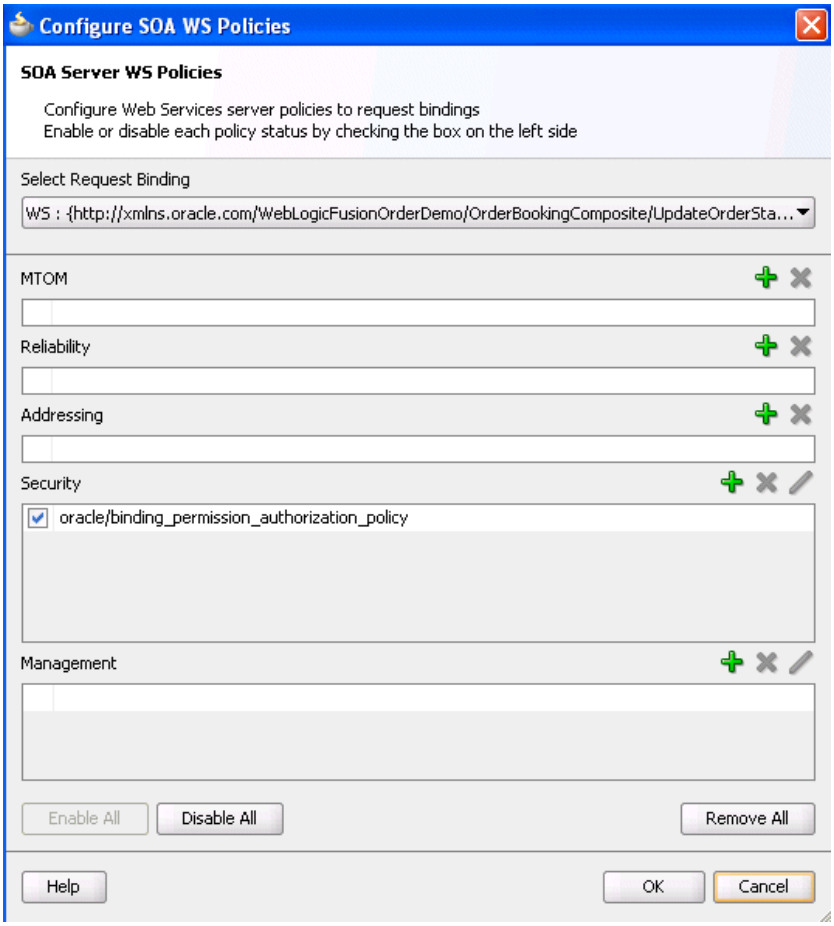
Figure 39–2 Security Policies



5. Place your cursor over a policy name to display a description of policy capabilities.
6. Select the type of policy to attach.
7. Click **OK**.

   You are returned to the Configure SOA WS Policies dialog shown in Figure 39–3. The attached security policy displays in the **Security** section.

Figure 39–3 Attached Security Policy



8. If necessary, add additional policies.

   You can temporarily disable a policy by deselecting the checkbox to the left of the name of the attached policy. This action does *not* detach the policy.

9. To detach a policy, click the **Delete** icon.

10. When complete, click **OK** in the Configure SOA WS Policies dialog.

    You are returned to the SOA Composite Editor.

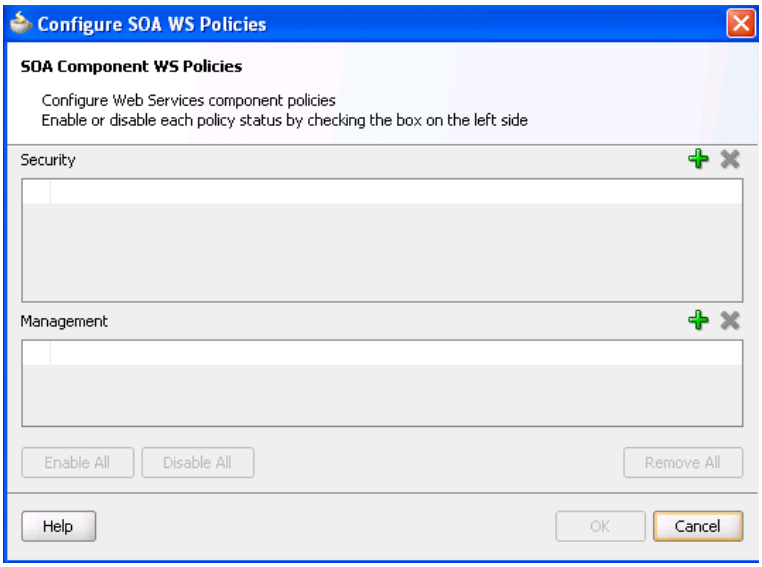
To attach a policy to a service component:

1. Right-click a service component.

2. Select **Configure Component WS Policies**.

   The Configure SOA WS Policies dialog shown in Figure 39–4 appears.

Figure 39–4 Configure SOA WS Policies Dialog



3. Click the **Add** icon for the type of policy to attach.
   - **Security**
   - **Management**

   The dialog for your selection appears.
4. Select the type of policy to attach.
5. Click **OK**.
6. If necessary, add additional policies.
7. When complete, click **OK** in the Configure SOA WS Policies dialog.

For information about attaching policies during runtime in Oracle Enterprise Manager Fusion Middleware Control, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

39.2.2 How to Override Policy Configuration Property Values

Your environment may include multiple clients or servers with the same policies. However, each client or server may have their own specific policy requirements. You can override the policy property values based on your runtime requirements.

39.2.2.1 Overriding Client Configuration Property Values

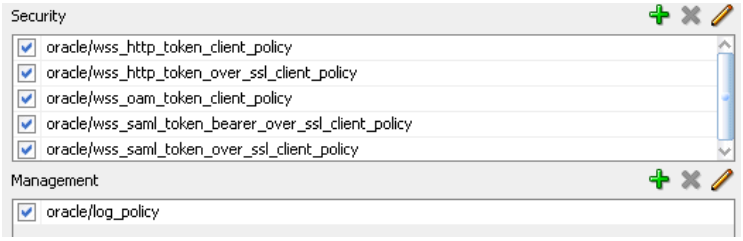
You can override the default values of client policy configuration properties on a per client basis without creating new policies for each client. In this way, you can override client policies that define default configuration values and customize those values based on your runtime requirements.

1. Right-click one of the following binding components:
   - A service binding component in the **Exposed Services** swimlane, and select **For Callback**.
   - A reference binding component in the **External References** swimlane, and select **For Request**.

2. Go to the **Security** and **Management** sections. These instructions assume you previously attached policies in these sections.

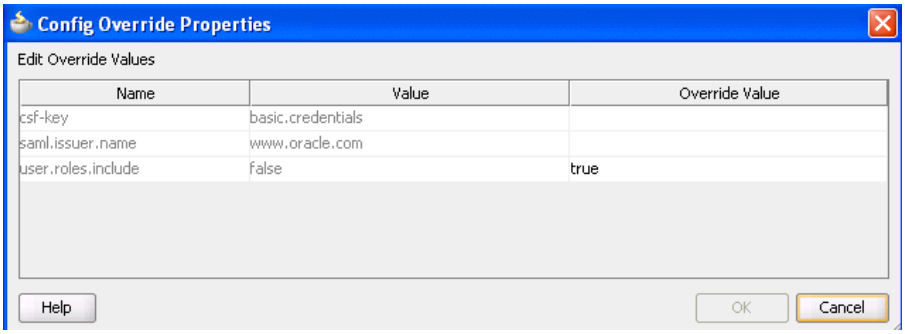
   Note that the **Edit** icon is enabled for both sections. Figure 39–5 provides details.

Figure 39–5 Client Policy Selection



3. Click the **Edit** icon. Note that regardless of which policies you select, the property names, values, and overridden values display for *all* of your attached client policies.
4. In the **Override Value** column, enter a value to override the default value shown in the **Value** column. Figure 39–6 provides details.

Figure 39–6 Client Policy Override Value



5. Click **OK** to exit the Config Override Properties dialog.
6. Click **OK** to exit the Configure SOA WS Policies dialog.
7. Click the **Source** button in the SOA Composite Editor.

   The overriding value is reflected with the `property name` attribute in the `composite.xml` file, as shown in Example 39–1.

Example 39–1 Client Policy Override Value in composite.xml File

```
<binding.ws port="http://xmlns.oracle.com/Application26_
jws/Project1/BPELProcess1#wsdl.endpoint(bpelprocess1_client_
ep/BPELProcess1Callback_pt)">
        <wsp:PolicyReference URI="oracle/wss_http_token_client_policy"
                             orawsp:category="security"
                             orawsp:status="enabled"/>
        <wsp:PolicyReference URI="oracle/wss_http_token_over_ssl_client_policy"
                             orawsp:category="security"
                             orawsp:status="enabled"/>
        <wsp:PolicyReference URI="oracle/wss_oam_token_client_policy"
                             orawsp:category="security"
                             orawsp:status="enabled"/>
        <wsp:PolicyReference URI="oracle/wss_saml_token_bearer_over_ssl_client_
```

```
policy"
                            orawsp:category="security"
                            orawsp:status="enabled"/>
        <wsp:PolicyReference URI="oracle/wss_saml_token_over_ssl_client_policy"
                            orawsp:category="security"
                            orawsp:status="enabled"/>
        <wsp:PolicyReference URI="oracle/log_policy"
                            orawsp:category="management"
                            orawsp:status="enabled"/>
<property name="user.roles.include" type="xs:string" many="false">true</property>
      </binding.ws>
```

For more information about overriding policy settings, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

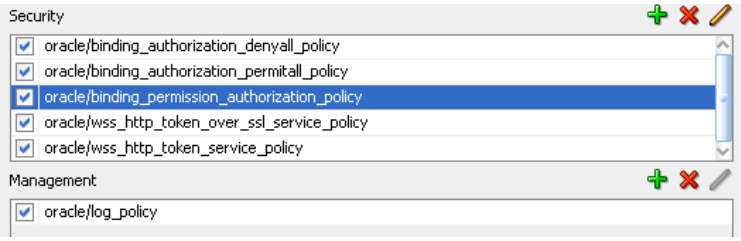
39.2.2.2 Overriding Server Configuration Property Values

You can override the default values of server policy configuration properties on a per server basis without creating new policies for each server. In this way, you can override server policies that define default configuration values and customize those values based on your runtime requirements.

1. Right-click one of the following binding components:
   - A service binding component in the **Exposed Services** swimlane, and select **For Request**.
   - A reference binding component in the **External References** swimlane, and select **For Callback**.
2. Go to the **Security** or **Management** section. These instructions assume you previously attached policies in these sections.

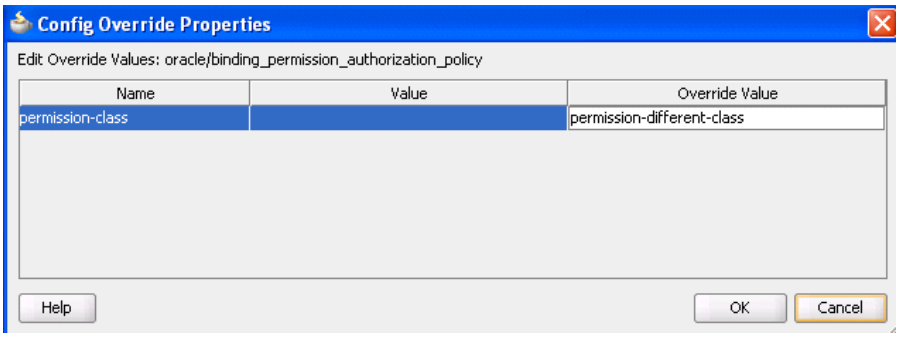
   Note that the **Edit** icon is *not* enabled by default for both sections. You must explicitly select a policy to enable this icon. This is because you can override fewer property values for the server. Figure 39–7 provides details.

Figure 39–7 Server Policy Selection



3. Select an attached policy that permits you to override its value, and click the **Edit** icon.
4. In the **Override Value** column, enter a value to override the default value shown in the **Value** column. Figure 39–8 provides details. If the policy store is unavailable, the words `no property store found in the store` display in red in the **Value** column.

Figure 39–8 Server Policy Override Value



5. Click **OK** to exit the Config Override Properties dialog.
6. Click **OK** to exit the Configure SOA WS Policies dialog.
7. Click the **Source** button in the SOA Composite Editor.

   The overriding value is reflected with the `OverrideProperty` attribute in the `composite.xml` file, as shown in Example 39–2.

Example 39–2 Server Policy Override Value in composite.xml File

```
<wsp:PolicyReference URI="oracle/binding_authorization_denyall_policy"
                     orawsp:category="security" orawsp:status="enabled"/>
    <wsp:PolicyReference URI="oracle/binding_authorization_permitall_policy"
                     orawsp:category="security" orawsp:status="enabled"/>
    <wsp:PolicyReference URI="oracle/binding_permission_authorization_policy"
                     orawsp:category="security" orawsp:status="enabled">
      <orawsp:OverrideProperty orawsp:name="permission-class"
                               orawsp:value="permission-different-class"/>
    </wsp:PolicyReference>
```

For more information about overriding policy settings, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

40

Deploying SOA Composite Applications

This chapter describes the deployment life cycle of SOA composite applications. Deployment prerequisite, packaging, preparation, and configuration tasks are described. Procedures for deploying composites with Oracle JDeveloper and scripting tools and creating configuration plans for moving SOA composite applications to and from different environments are also provided.

This chapter includes the following sections:

See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for instructions on deploying SOA composite applications from Oracle Enterprise Manager Fusion Middleware Control and *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference* for instructions on deploying SOA composite applications with the WebLogic Scripting Tool (WLST) utility.

40.1 Introduction to Deployment

This chapter describes the following deployment life cycle topics:

- Deployment prerequisites
- Packaging details
- Anatomy of a composite
- Target environment preparation
- Target environment configuration tasks
- Composite deployment
- Postdeployment configuration tasks

- Testing and troubleshooting composite applications

For more information about the deployment life cycle, see *Oracle Fusion Middleware Administrator's Guide*.

40.2 Deployment Prerequisites

This section describes the basic prerequisites required for creating and deploying a SOA composite application.

40.2.1 Creating the Oracle SOA Suite Schema

Oracle SOA Suite components require schemas that must be installed in the Oracle or Microsoft SQL Server database. You create and load these schemas in your database with the Repository Creation Utility (RCU). For information about installing and configuring your schemas, see *Oracle Fusion Middleware Installation Guide for Oracle SOA Suite and Oracle Business Process Management Suite* and *Oracle Fusion Middleware Repository Creation Utility User's Guide*.

40.2.2 Creating a SOA Domain

After installation, you use the Oracle Fusion Middleware Configuration Wizard to create and configure a new Oracle WebLogic Server domain, and choose products such as Oracle SOA Suite to configure in that domain. This new domain contains the administration server and other managed servers, depending on the products you choose to configure.

For more information, see *Oracle Fusion Middleware Installation Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

40.2.3 Configuring a SOA Cluster

You can deploy a SOA composite application into a clustered environment. For more information on creating and configuring a clustered environment, see *Oracle Fusion Middleware High Availability Guide*.

40.3 Understanding the Packaging Impact

You can separately package all required artifact files within a project of a SOA composite application into a SOA archive (SAR) JAR file though use of the following tools:

- Oracle JDeveloper

  During deployment on the Deployment Action page, you select the **Deploy to SAR** option. For more information, see Section 40.7.1.1.3, "Deploying the Profile."

- `ant` scripts

  Use the `ant-sca-package` script to package your artifacts. For more information, see Section 40.7.5.2.3, "Packaging a SOA Composite Application into a Composite SAR File."

- WLST commands

  Use the `sca_package` script to package your artifacts. For more information, see *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.

A SAR file is a special JAR file that requires a prefix of `sca_` (for example, `sca_HelloWorld_rev1.0.jar`).

In addition, when you deploy a SOA composite application with the **Deploy to Application Server** option on the Deployment Action page in Oracle JDeveloper, all required artifact files within a project are automatically packaged into one of the following files:

- A self-contained JAR file (for single SOA composite applications)

  For more information about self-contained composites, see Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper" and Section 40.7.2, "Deploying Multiple SOA Composite Applications in Oracle JDeveloper."

- A ZIP file of multiple SOA composite applications that share metadata with one another

  You can deploy and use shared metadata across SOA composite applications. Shared metadata is deployed to the SOA Infrastructure on the application server as a metadata service (MDS) archive JAR file. The archive file contains all shared resources. For more information, see Section 40.7.3, "Deploying and Using Shared Metadata Across SOA Composite Applications in Oracle JDeveloper."

40.4 Anatomy of a Composite

When you deploy a SOA composite application in Oracle JDeveloper, the composite is packaged in a JAR file (for a single composite application) or a ZIP file (for multiple SOA composite applications). These files can include the following artifacts:

- Binding components and service components.
- References to B2B agreements, Oracle Web Service Manager (OWSM) policies, and human workflow task flows.
- Metadata such as WSDL and XSD files. All shared metadata is deployed to an existing SOA Infrastructure partition on the server. This metadata is deployed under the `/apps` namespace. When you refer to this artifact in Oracle JDeveloper using a SOA-MDS connection, the URL is prefixed with `oramds`.

40.5 Preparing the Target Environment

The target environment is the SOA Infrastructure environment to which you want to deploy your SOA composite application. This is typically a development, test, or production environment. Depending upon the components, identity service provider, and security policies you are using in your composite application, additional configuration steps may be required as you move your application from one target environment to another. This section describes these tasks.

40.5.1 Creating Data Sources and Queues

A JDBC data source is an object bound to the JNDI tree that includes a pool of JDBC connections. Applications can look up a data source on the JNDI tree and then reserve a database connection from the data source. You create queues in which to enqueue outgoing messages or dequeue incoming messages. The Oracle JCA adapters listed in Table 40–1 require JDBC data sources and queues to be configured before deployment.

Table 40–1 Oracle JCA Adapter Tasks

Adapter	Configuration Task	See Section...
Database adapter	JDBC data source	"Deployment" of *Oracle Fusion Middleware User's Guide for Technology Adapters*
AQ adapter	JDBC data source	"Configuring the Data Sources in the Oracle WebLogic Server Administration Console" of *Oracle Fusion Middleware User's Guide for Technology Adapters*
JMS adapter	Queue	"Using the Adapter Configuration Wizard to Configure Oracle JMS Adapter" of *Oracle Fusion Middleware User's Guide for Technology Adapters*

40.5.1.1 Script for Creation of JMS Resource and Redeployment of JMS Adapter

Example 40–1 provides a script for creating the JMS resource and redeploying the JMS adapter.

> **Note:** This script is for demonstration purposes. You may need to modify this script based on your environment.

Example 40–1 Script for Creation of JMS Resource and Redeployment of JMS Adapter

```
# lookup the JMSModule
    jmsSOASystemResource = lookup("SOAJMSModule","JMSSystemResource")

    jmsResource = jmsSOASystemResource.getJMSResource()

    cfbean = jmsResource.lookupConnectionFactory('DemoSupplierTopicCF')
    if cfbean is None:
        print "Creating DemoSupplierTopicCF connection factory"
        demoConnectionFactory =
 jmsResource.createConnectionFactory('DemoSupplierTopicCF')
        demoConnectionFactory.setJNDIName('jms/DemoSupplierTopicCF')
        demoConnectionFactory.setSubDeploymentName('SOASubDeployment')

    topicbean = jmsResource.lookupTopic('DemoSupplierTopic')
    if topicbean is None:
        print "Creating DemoSupplierTopic jms topic"
        demoJMSTopic = jmsResource.createTopic("DemoSupplierTopic")
        demoJMSTopic.setJNDIName('jms/DemoSupplierTopic')
        demoJMSTopic.setSubDeploymentName('SOASubDeployment')

try:
    save()
    # activate the changes
    activate(block="true")
    print "jms topic and factory for SOA Fusion Order Demo successfully created"
except:
    print "Error while trying to save and/or activate!!!"
    dumpStack()

print "Creating jms adapter connection factory information"
try:
     redeploy('JmsAdapter', '@deployment.plan@', upload='true', stageMode='stage')

except:
    print "Error while modifying jms adapter connection factory"
```

40.5.1.2 Script for Creation of the Database Resource and Redeployment of the Database Adapter

Example 40–2 provides a script for creating the database resource and redeploying the database adapter.

> **Note:** This script is for demonstration purposes. You may need to modify this script based on your environment.

Example 40–2 Script for Creation of the Database Resource and Redeployment of the Database Adapter

```
import os
connect(userName,passWord,'t3://'+wlsHost+':'+adminServerListenPort)
edit()
startEdit()

soaJDBCSystemResource1 = create('DBAdapterTestDataSource',"JDBCSystemResource")
soaJDBCResource1 = soaJDBCSystemResource1.getJDBCResource()
soaJDBCResource1.setName('DBAdapterDataSource')

soaConnectionPoolParams1 = soaJDBCResource1.getJDBCConnectionPoolParams()
soaConnectionPoolParams1.setTestTableName("SQL SELECT 1 FROM DUAL")

soaConnectionPoolParams1.setInitialCapacity(10)
soaConnectionPoolParams1.setMaxCapacity(100)

soaDataSourceParams1 = soaJDBCResource1.getJDBCDataSourceParams()
soaDataSourceParams1.addJNDIName('jdbc/dbSample')
soaDriverParams1 = soaJDBCResource1.getJDBCDriverParams()
soaDriverParams1.setUrl('jdbc:oracle:thin:@'+db_host_name+':'+db_port+':'+db_sid)
soaDriverParams1.setDriverName('oracle.jdbc.xa.client.OracleXADataSource')
soaDriverParams1.setPassword('my_password')

soaDriverProperties1 = soaDriverParams1.getProperties()
soaProperty1 = soaDriverProperties1.createProperty("user")
soaProperty1.setValue('scott')

varSOAServerTarget = '/Servers/'+serverName
soaServerTarget = getMBean(varSOAServerTarget)

soaJDBCSystemResource1.addTarget(soaServerTarget)

dumpStack()

try :

save()

activate(block="true")

except:
    print "Error while trying to save and/or activate!!!"
    dumpStack()

print "Creating DB adapter resource  information"
try:
     redeploy('DBAdapter', '@deployment.plan@', upload='true', stageMode='stage')
```

```
except:
    print "Error while modifying db adapter connection factory"
```

40.5.2 Creating Connection Factories and Connection Pooling

The Oracle JCA adapters are deployed as JCA 1.5 resource adapters in an Oracle WebLogic Server container. Adapters are packaged as Resource Adapter Archive (RAR) files using a JAR format. When adapters are deployed, the RAR files are used and the adapters are registered as connectors with the Oracle WebLogic Server or middle-tier platform. The RAR file contains the following:

- The `ra.xml` file, which is the deployment descriptor XML file containing deployment-specific information about the resource adapter
- Declarative information about the contract between Oracle WebLogic Server and the resource adapter

Adapters also package the `weblogic-ra.xml` template file, which defines the endpoints for connection factories.

For information about creating connection factories and connection pools, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

40.5.3 Enabling Security

If you are using an identity service provider with human workflow or attaching authentication and authorization policies, you must perform additional setup tasks.

- Identity service provider for human workflow

  By default, the identity service uses the embedded LDAP server in Oracle WebLogic Server as the default authentication provider. If you are using human workflow, you can configure Oracle WebLogic Server to use an alternative identity service provider, such as Oracle Internet Directory, Microsoft Active Directory, or Sun iPlanet. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*. Note that the embedded LDAP server is not supported in clustered environments.

- Authentication provider (OWSM policies)

  Policies that use certain types of tokens (for example, the username, X.509, and SAML tokens) require an authentication provider. For information about selecting and configuring an authentication provider, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

- Authorization provider (OWSM policies)

  After a user is authenticated, you must verify that the user is authorized to access a web service with an authorization policy. You can create an authorization policy with several types of assertion templates. For information about authorization policies and which resources to protect, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

40.5.4 Deploying Trading Partner Agreements and Task Flows

If you are using Oracle B2B or a human task, you must perform additional setup tasks.

- Deploying trading partner agreements

  A trading partner agreement defines the terms that enable two trading partners, the initiator and the responder, to exchange business documents. It identifies the

trading partners, trading partner identifiers, document definitions, and channels. You must deploy the agreement from the design-time repository to the run-time repository. For more information, see *Oracle Fusion Middleware User's Guide for Oracle B2B*.

- Deploying the task flow

  You must deploy the task flow in order to use it in Oracle BPM Worklist.

40.5.5 Creating an Application Server Connection

To deploy a SOA composite application that does not share metadata with another composite, use the Create Application Server Connection wizard to create an application server connection. For more information, see Section 40.7.1.1.1, "Creating an Application Server Connection."

40.5.6 Creating a SOA-MDS Connection

To deploy a SOA composite application that shares metadata with other composites, use the Create SOA-MDS Connection wizard to create a connection to a database-based MDS server. For more information, see Section 40.7.3.2.1, "Creating a SOA-MDS Connection."

40.6 Customizing Your Application for the Target Environment Prior to Deployment

Not all customization tasks must be manually performed as you move to and from development, test, and production environments. This section describes how to use a configuration plan to automatically configure your SOA composite application for the next target environment.

40.6.1 Customizing SOA Composite Applications for the Target Environment

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 with WebLogic Scripting Tool (WLST) commands. During process deployment, the configuration plan searches the SOA project for values that must be replaced to adapt the project to the next target environment.

40.6.1.1 Introduction to Configuration Plans

This section provides an overview of creating and attaching a configuration plan:

- You create and edit a configuration plan file in which you can replace the following attributes and properties:
  - Any composite, service component, reference, service, and binding properties in the SOA composite application file (`composite.xml`)
  - Attribute values for bindings (for example, the location for `binding.ws`)
  - `schemaLocation` attribute of an import in a WSDL file
  - `location` attribute of an include in a WSDL file
  - `schemaLocation` attribute of an include, import, and redefine in an XSD file

  - Any properties in JCA adapter files
  - Modify and add policy references for the following:
    * Service component
    * Service and reference binding components

    Note: The configuration plan does not alter XSLT artifacts in the SOA composite application. If you want to modify any XSL, do so in the XSLT Mapper. Using a configuration plan is not useful. For example, you cannot change references in XSL using the configuration plan file. Instead, they must be changed manually in the XSLT Mapper in Oracle JDeveloper when moving to and from test, development, and production environments. This ensures that the XSLT Mapper opens without any issues in design time. However, leaving the references unchanged does not impact runtime behavior.

- You attach the configuration plan file to a SOA composite application JAR file or ZIP file (if deploying a SOA bundle) during deployment with one of the following tools:
  - Oracle JDeveloper

    For more information, see Section 40.7.1.1.3, "Deploying the Profile."
  - `ant` scripts

    For more information, see Section 40.7.5.2.4, "Deploying a SOA Composite Application."
  - WLST commands

    For more information, see *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.
- During deployment, the configuration plan file searches the `composite.xml`, WSDL, and XSD files in the SOA composite application JAR or ZIP file for values that must be replaced to adapt the project to the next target environment.

40.6.1.2 Introduction to a Configuration Plan File

The following example shows a configuration plan in which you modify the following:

- An `inFileFolder` property for composite `FileAdaptorComposite` is replaced with `mytestserver/newinFileFolder`.
- A hostname (`myserver17`) is replaced with `test-server` and port `8888` is replaced with `8198` in the following locations:
  - All import WSDLs
  - All reference `binding.ws` locations

The `composite.xml` file looks as shown in Example 40–3:

Example 40–3 composite.xml File

```
<composite .....>
  <import namespace="http://example.com/hr/"
 location="http://myserver17.us.oracle.com:8888/hrapp/HRAppService?WSDL"
 importType="wsdl"/>
  <service name="readPO">
```

```
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/file/readPO/#wsdl.interface(Read
_ptt)"/>
    <binding.jca config="readPO_file.jca"/>
    <property name="inFileFolder" type="xs:string" many="false"
 override="may">/tmp/inFile</property>
  </service>
  <reference name="HRApp">
    <interface.wsdl
 interface="http://example.com/hr/#wsdl.interface(HRAppService)"/>
    <binding.ws
port="http://example.com/hr/#wsdl.endpoint(HRAppService/HRAppServiceSoapHttpPort)"
 location="http://myserver17.us.oracle.com:8888/hrapp/HRAppService?WSDL"/>
    <binding.java serviceName="{http://example.com/hr/}HRAppService"
 registryName="HRAppCodeGen_JBOServiceRegistry"/>
  </reference>
</composite>
```

The configuration plan file looks as shown in Example 40–4.

Example 40–4 Configuration Plan File

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAConfigPlan
 xmlns:jca="http://platform.integration.oracle/blocks/adapter/fw/metadata"
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
 xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
 xmlns:edl="http://schemas.oracle.com/events/edl"
 xmlns="http://schemas.oracle.com/soa/configplan">
  <composite name="FileAdaptorComposite">
    <service name="readPO">
      <binding type="*">
        <property name="inFileFolder">
          <replace>/mytestserver/newinFileFolder</replace>
        </property>
      </binding>
    </service>
  </composite>
  <!-- For all composite replace host and port in all imports wsdls -->
  <composite name="*">
    <import>
      <searchReplace>
        <search>myserver17</search>
        <replace>test-server</replace>
      </searchReplace>
      <searchReplace>
        <search>8888</search>
        <replace>8198</replace>
      </searchReplace>
    </import>
    <reference name="*">
      <binding type="ws">
        <attribute name="location">
          <searchReplace>
            <search>myserver17</search>
            <replace>test-server</replace>
          </searchReplace>
          <searchReplace>
            <search>8888</search>
            <replace>8198</replace>
```

```
          </searchReplace>
        </attribute>
      </binding>
    </reference>
  </composite>
</SOAConfigPlan>
```

A policy is replaced if a policy for the same URI is available. Otherwise, it is added. This is different from properties, which are modified, but not added.

40.6.1.3 Introduction to Use Cases for a Configuration Plan

The following steps provide an overview of how to use a configuration plan when moving from development to testing environments:

1. User A creates SOA composite application Foo.
2. User A deploys Foo to a development server, fixes bugs, and refines the process until it is ready to test in the staging area.
3. User A creates and edits a configuration plan for Foo, which enables the URLs and properties in the application to be modified to match the testing environment.
4. User A deploys Foo to the testing server using Oracle JDeveloper or a series of command-line scripts (can be WLST-based). The configuration plan created in Step 3 modifies the URLs and properties in Foo.
5. User A deploys SOA composite application Bar in the future and applies the same plan during deployment. The URLs and properties are also modified.

The following steps provide an overview of how to use a configuration plan when creating environment-independent processes:

> **Note:** This use case is useful for users that have their own development server and a common development and testing server if they share development of the same process. Users that share the same deployment environment (that is, the same development server) may not find this use case as useful.

1. User A creates SOA composite application Foo.
2. User A deploys Foo to their development server, fixes bugs, and refines the process until it is ready to test in the staging area.
3. User A creates a configuration plan for Foo, which enables the URLs and properties in the process to be modified to match the settings for User A's environment.
4. User A checks in Foo and the configuration plan created in Step 3 to a source control system.
5. User B checks out Foo from source control.
6. User B makes a copy of the configuration plan to match their environment and applies the new configuration plan onto Foo's artifacts.
7. User B imports the application into Oracle JDeveloper and makes several changes.
8. User B checks in both Foo and configuration plan B (which matches user B's environment).
9. User A checks out Foo again, along with both configuration plans.

40.6.1.4 How to Create a Configuration Plan in Oracle JDeveloper

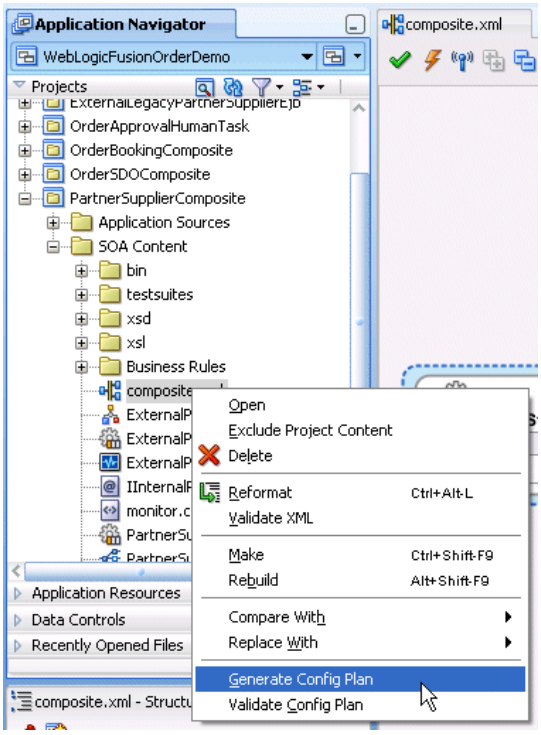
This section describes how to create and use a configuration plan. In particular, this section describes the following:

- Creating and editing a configuration plan
- Attaching the configuration plan to a SOA composite application JAR file
- Validating the configuration plan
- Deploying the SOA composite application JAR or ZIP file in which the configuration plan is included

To create a configuration plan in Oracle JDeveloper:

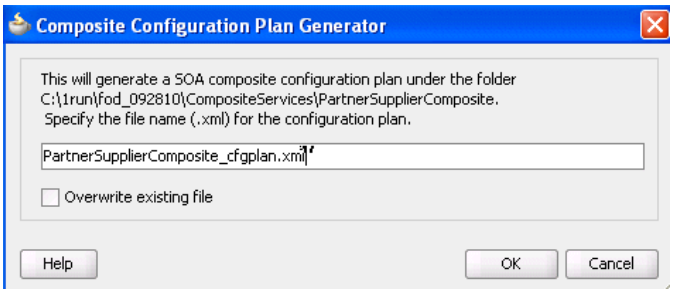
1. Open Oracle JDeveloper.
2. Right-click the **composite.xml** file of the project in which to create a configuration plan, and select **Generate Config Plan**. Figure 40–1 provides details.

Figure 40–1 Generate a Configuration Plan



The Composite Configuration Plan Generator dialog appears, as shown in Figure 40–2.

Figure 40–2 Composite Configuration Plan Generator Dialog



3. Create a configuration plan file for editing, as shown in Table 40–2.

Table 40–2 Generate a Configuration Plan

Field	Description
Specify the file name (.xml) for the configuration plan	Enter a specific name or accept the default name for the configuration plan. The file is created in the directory of the project and packaged with the SOA composite application JAR or ZIP file. **Note:** During deployment, you can specify a different configuration file when prompted in the Deploy Configuration page of the deployment wizard.
Overwrite existing file	Click to overwrite an existing configuration plan file with a different file in the project directory.

4. Click **OK**.

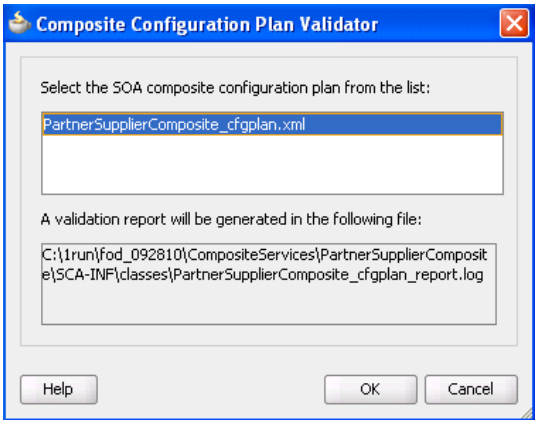
   This creates and opens a single configuration plan file for editing, similar to that shown in Example 40–4. You can modify URLs and properties for the `composite.xml`, WSDL, and schema files of the SOA composite application. Figure 40–3 provides details.

Figure 40–3 Configuration Plan Editor

5. Add values for server names, port numbers, and so on to the existing syntax. You can also add replacement-only syntax when providing a new value. You can add multiple search and replacement commands in each section.
6. From the **File** menu, select **Save All**.
7. Above the editor, click the **x** to the right of the file name to close the configuration plan file.
8. Right-click the **composite.xml** file again, and select **Validate Config Plan**.

   The Composite Configuration Plan Validator appears, as shown in Figure 40–4.

Figure 40–4 Validate the Configuration Plan



9. Select the configuration plan to validate. This step identifies all search and replacement changes to be made during deployment. Use this option for debugging only.
10. Note the directory in which a report describing validation results is created, and click **OK**.

    The Log window in Oracle JDeveloper indicates if validation succeeded and lists all search and replacement commands to perform during SOA composite application deployment. This information is also written to the validation report.

> **Note:** The old `composite.xml`, WSDL, and XSD files are not replaced with files containing the new values for the URLs and properties appropriate to the next environment. Replacement occurs only when the SOA composite application is deployed.

11. Deploy the SOA composite application by following the instructions in one of the following sections:
    - Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper"
    - Section 40.7.2, "Deploying Multiple SOA Composite Applications in Oracle JDeveloper"
    - Section 40.7.3, "Deploying and Using Shared Metadata Across SOA Composite Applications in Oracle JDeveloper"

    During deployment, the Deploy Configuration page shown in Step 4 of Section 40.7.1.1.3, "Deploying the Profile" prompts you to select the configuration plan to include in the SOA composite application archive.

12. Select the configuration plan to include with the SOA composite application.
13. Click **OK**.

40.6.1.5 How to Create a Configuration Plan with the WLST Utility

As an alternative to using Oracle JDeveloper, you can use the WLST command line utility to perform the following configuration plan management tasks:

- Generate a configuration plan for editing:

```
sca_generatePlan(configPlan, sar, composite, overwrite, verbose)
```

- Attach the configuration plan file to the SOA composite application JAR file:

```
sca_attachPlan(sar, configPlan, overwrite, verbose)
```

- Validate the configuration plan:

```
sca_validatePlan(reportFile, configPlan, sar, composite, overwrite, verbose)
```

- Extract a configuration plan packaged with the JAR file for editing:

```
sca_extractPlan(sar, configPlan, overwrite, verbose)
```

For information on how to use these commands, see *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.

40.6.1.6 How to Attach a Configuration Plan with ant Scripts

As an alternative to using Oracle JDeveloper, you can use `ant` scripts to attach the configuration plan file to the SOA composite application JAR or ZIP file during deployment. For instructions, see Section 40.7.5.2.4, "Deploying a SOA Composite Application."

40.7 Deploying SOA Composite Applications

This section describes how to deploy the following types of SOA composite applications.

- Deploying a single composite in Oracle JDeveloper
- Deploying multiple composites in Oracle JDeveloper
- Deploying and using shared metadata in Oracle JDeveloper
- Deploying an existing SOA archive in Oracle JDeveloper
- Managing SOA composite applications with WLST and `ant` scripts
- Deploying from Oracle Enterprise Manager Fusion Middleware Control
- Deploying SOA composite applications to a cluster

40.7.1 Deploying a Single SOA Composite in Oracle JDeveloper

Oracle JDeveloper requires the use of profiles for SOA projects and applications to be deployed to Oracle WebLogic Server.

40.7.1.1 How to Deploy a Single SOA Composite

This section describes how to deploy a single SOA composite application with Oracle JDeveloper.

40.7.1.1.1 Creating an Application Server Connection

You must create a connection to the application server to which to deploy a SOA composite application. The following 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**.
2. In the **General** list, select **Connections**.
3. Select **Application Server Connection**, and click **OK**.

   The Name and Type page appears.
4. In the **Connection Name** field, enter a name for the connection.
5. In the **Connection Type** list, select **WebLogic 10.3** to create a connection to Oracle WebLogic Server.
6. Click **Next**.

   The Authentication page appears.
7. In the **Username** field, enter the user authorized for access to the application server.
8. In the **Password** field, enter the password for this user.
9. Click **Next**.

   The Configuration page appears.
10. In the **Weblogic Hostname (Administration Server)** field, enter the host on which the Oracle WebLogic Server is installed.
11. In the **Port** and **SSL Port** fields, enter the appropriate port values or accept the default values.
12. If you want to use SSL, select the **Always use SSL** checkbox. Table 40–3 describes what occurs when you select this checkbox.

Table 40–3 Deployment to HTTPS and HTTP Servers

If This Checkbox Is...	Then...
Selected	An HTTPS server URL must exist to deploy the composite with SSL. Otherwise, deployment fails. If the server has only an HTTP URL, deployment also fails. This option enables you to ensure that SSL deployment must *not* go through a non-SSL HTTP URL, and must only go through an HTTPS URL.
Not selected	An HTTP server URL must exist to deploy to a non-SSL environment. Otherwise, deployment fails. If the server has both HTTPS and HTTP URLs, deployment occurs through a non-SSL connection. This option enables you to force a non-SSL deployment from Oracle JDeveloper, even though the server is SSL-enabled.

13. In the **WebLogic Domain** field, enter the Oracle SOA Suite domain. For additional details about specifying domains, click **Help**.
14. Click **Next**.

15. Click **Test Connection** to test your server connection.
16. If the connection is successful, click **Finish**. Otherwise, click **Back** to make corrections in the previous dialogs. Even if the connection test is unsuccessful, a connection is created.

40.7.1.1.2 Optionally Creating a Project Deployment Profile

A required deployment profile is automatically created for your project. The application profile includes the JAR files of your SOA projects. If you want, you can create additional profiles.

To create a project deployment profile:

1. In the Application Navigator, right-click the SOA project.
2. Select **Project Properties**.

   The Project Properties dialog appears.
3. Click **Deployment**.
4. Click **New**.

   The Create Deployment Profile dialog appears.
5. Enter the values shown in Table 40–4.

Table 40–4 Create Deployment Profile Dialog Fields and Values

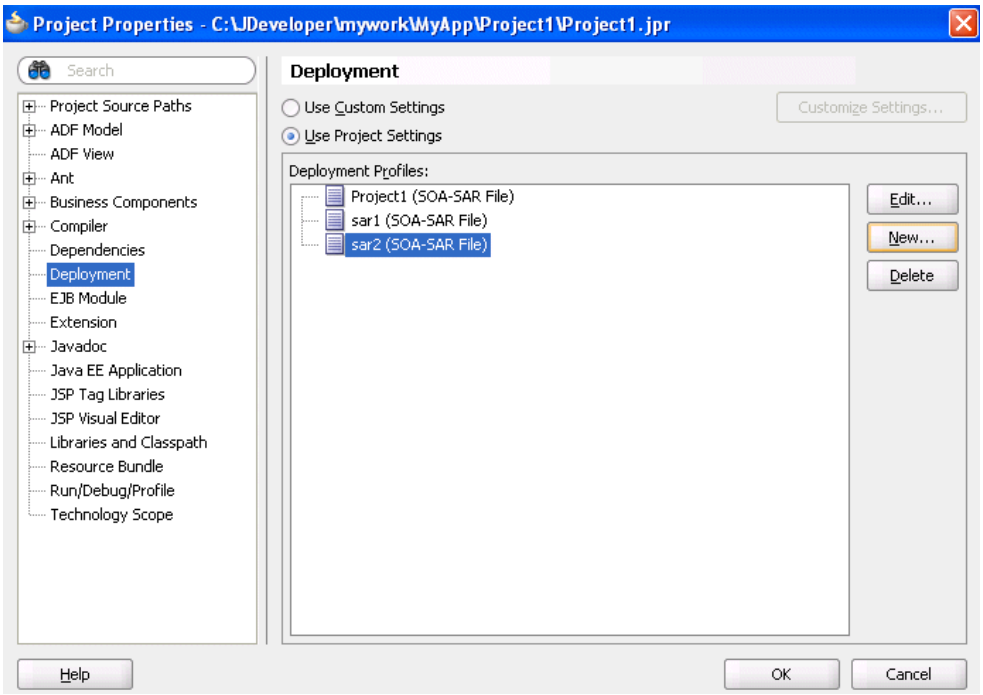
Field	Description
Archive Type	Select **SOA-SAR File**. A SAR is a deployment unit that describes the SOA composite application. The SAR packages service components such as BPEL processes, business rules, human tasks, and Oracle Mediator routing services into a single application. The SAR file is analogous 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 Oracle Mediator routing services).
Name	Enter a deployment profile name.

6. Click **OK**.

   The SAR Deployment Profile dialog appears.
7. Click **OK** to close the SAR Deployment Profile Properties dialog.

   The deployment profile shown in Figure 40–5 displays in the Project Properties dialog.

Figure 40–5 Project Profile



40.7.1.1.3 Deploying the Profile You now deploy the project profile to Oracle WebLogic Server. Deployment requires the creation of an application server connection. You can create a connection during deployment by clicking the **Add** icon in Step 10 or before deployment by following the instructions in Section 40.7.1.1.1, "Creating an Application Server Connection."

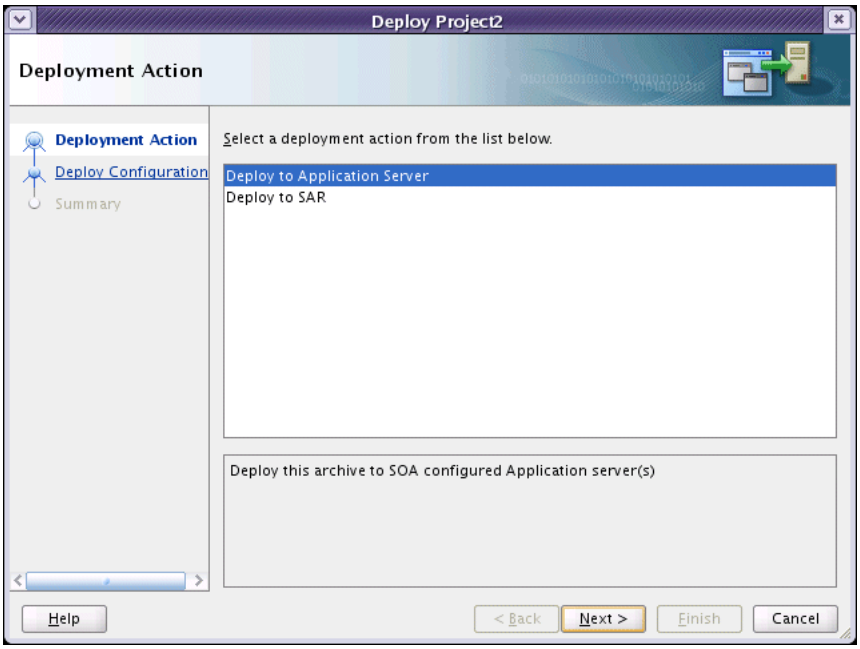
To deploy the profile:

1. In the Application Navigator, right-click the SOA project.
2. Select **Deploy** > ***project_name***.

   The value for ***project_name*** is the SOA project name.

   The Deployment Action page of the Deploy *Project_Name* wizard appears. Figure 40–6 provides an example.

Figure 40–6 Deployment Action Page



3. Select one of the following deployment options:

   - **Deploy to Application Server**

     Creates a JAR file for the selected SOA project and deploys it to an application server such as Oracle WebLogic Server.

   - **Deploy to SAR**

     Creates a SAR (JAR) file of the selected SOA project, but does *not* deploy it to an application server such as Oracle WebLogic Server. This option is useful for environments in which:

     – Oracle WebLogic Server may not be running, but you want to create the artifact JAR file.

     – You want to deploy multiple JAR files to Oracle WebLogic Server from a batch script. This option offers an alternative to opening all project profiles (which you may not have) and deploying them from Oracle JDeveloper.

   The page that displays differs based on your selection.

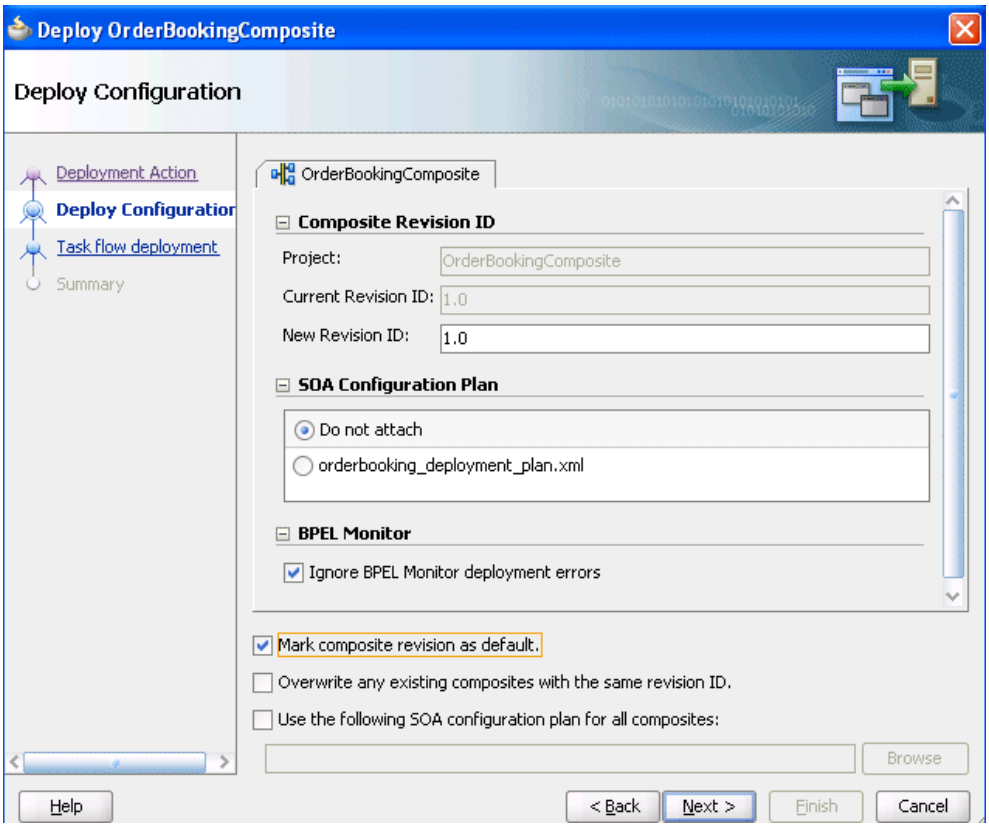
4. Select the deployment option appropriate for your environment.

Table 40–5 Deployment Target

If You Select...	Go to...
Deploy to Application Server	Step 4a
Deploy to SAR	Step 4b

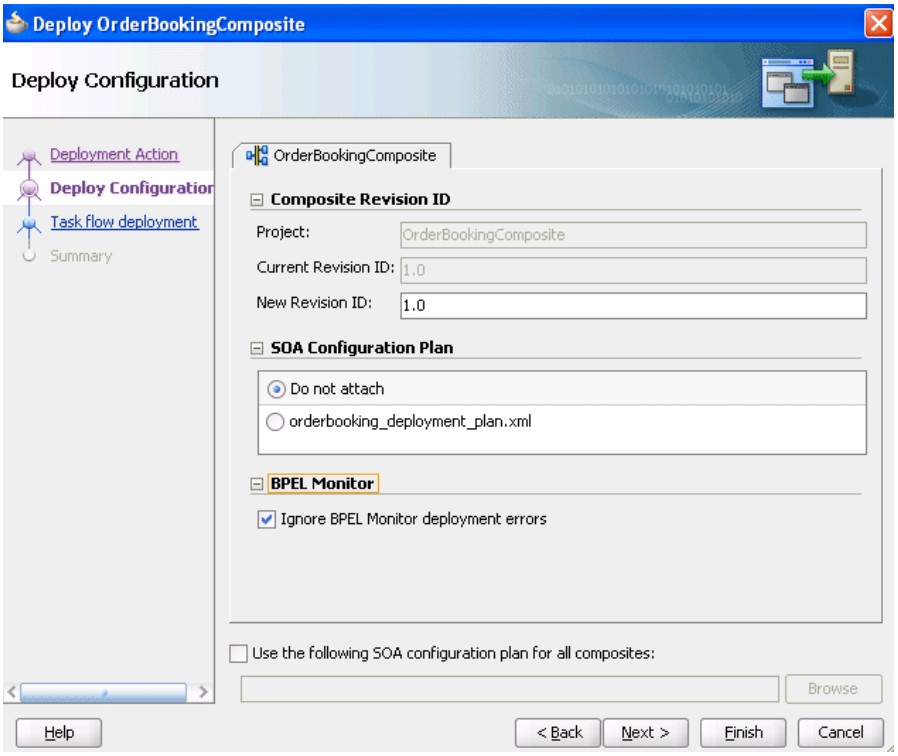
   a. View the Deploy Configuration page shown in Figure 40–7.

Figure 40–7 Deploy Configuration Page for Application Server Deployment



b. View the Deploy Configuration page shown in Figure 40–8.

Figure 40–8 Deploy Configuration Page for SAR Deployment



5. Provide values appropriate to your environment, as described in Table 40–6. If you selected to deploy to a server, additional fields display in the page.

Table 40–6 SOA Deployment Configuration Dialog

Field	Description
Composite Revision ID	Expand to display details about the project.
▪ **Project**	Displays the project name.
▪ **Current Revision ID**	Displays the current revision ID of the project.
▪ **New Revision ID**	Optionally change the revision ID of the SOA composite application.
SOA Configuration Plan	Expand to display details about the configuration plan. The configuration plan enables you to define the URL and property values to use in different environments. During process deployment, the configuration plan is used to search the SOA project for values that must be replaced to adapt the project to the next target environment.
▪ **Do not attach**	Select to not include a configuration plan with the SOA composite application JAR file. If you have not created a configuration plan, this field is disabled. This is the default selection.
▪ ***Configuration_plan*.xml**	Select the specific plan. A configuration plan must already exist in the SOA project for this selection to be available. See Section 40.6.1, "Customizing SOA Composite Applications for the Target Environment" for instructions on creating a configuration plan.
BPEL Monitor	Expand to display details about BPEL monitors.

Table 40–6 (Cont.) SOA Deployment Configuration Dialog

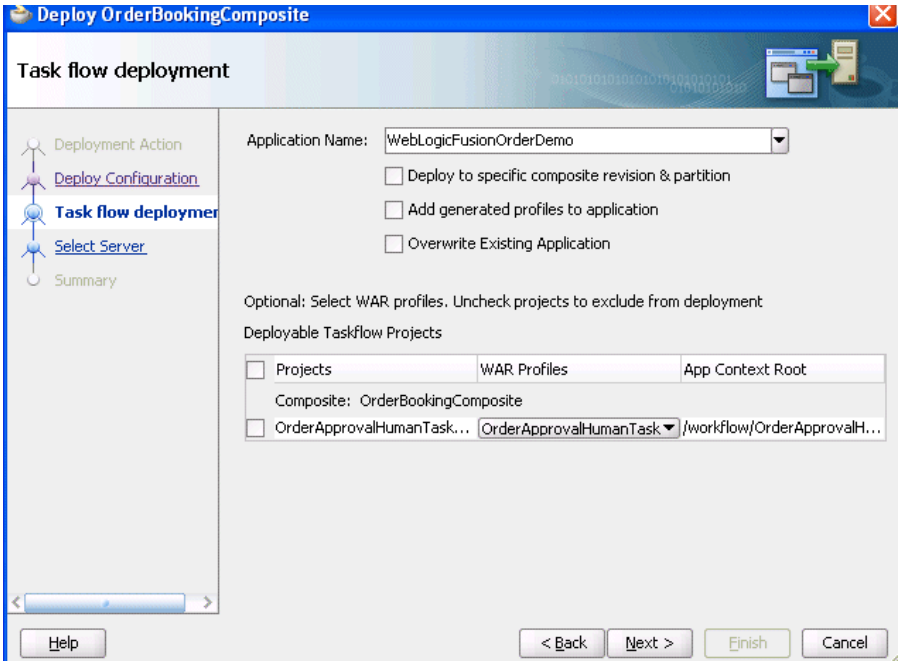
Field	Description
▪ **Ignore BPEL Monitor deployment errors** **Note:** This checkbox only appears if there is at least one `.monitor` file in the application.	Deselect this checkbox to display BPEL Monitor deployment errors. This checkbox corresponds to the `ignoreErrors` property in the `monitor.config` BPEL project file. This file defines runtime and deployment properties needed to connect with Oracle BAM Server to create the Oracle BAM data objects and dashboards. If Oracle BAM Server is unreachable, and `ignoreErrors` is set to `true`, deployment of the composite does not stop. If set to `false` and Oracle BAM Server is unavailable, deployment fails.
Mark composite revision as default	If you do not want the new revision to be the default, you can deselect this box. By default, a newly deployed composite revision is the default. This revision is instantiated when a new request comes in. The option only displays if you selected **Deploy to Application Server** on the Deployment Action page.
Overwrite any existing composites with the same revision ID	Select to overwrite any existing SOA composite application of the same revision value. The option only displays if you selected **Deploy to Application Server** on the Deployment Action page.
Use the following SOA configuration plan for all composites	Click **Browse** to select the same configuration plan to use for all composite applications. This option is used when deploying multiple composite applications.

6. Click **Next**.
7. If the SOA project you selected for deployment includes a task flow project defined for a human task, you are prompted with the Task Flow Deployment dialog, as shown in Figure 40–9.

   Otherwise, go to Step 10.

   You create or configure an Enterprise Resource Archive (EAR) file for the task flow forms of human tasks. The EAR file consists of a Web Resource Archive (WAR) profile that you select in the **Deployable Task Flow Projects** table of this dialog.

Figure 40–9 Task Flow Deployment Page



8. Provide values appropriate to your environment, as described in Table 40–7.

Table 40–7 Task Flow Deployment Dialog

Field	Description
Application Name	Select the EAR file to include in the deployment. This list displays all available EAR profiles in the current Oracle JDeveloper application. These EAR profiles are used as a template to create a new EAR profile to deploy based on the WAR profiles selected in the **Deployable Task Flow Projects** table. You can also enter any EAR profile name to deploy.
Deploy to specific composite revision & partition	Select to append the revision number of the composite to the EAR file name. If selected, this checkbox includes the composite revision in the EAR name, WAR profile, and context root. This option enables you to deploy an application specific to a composite revision.
Add generated profiles to application	Select to add the generated EAR profile to the current SOA composite application's EAR deployment profile list. The application may have to be saved to persist the generated EAR profile. Once the deployment profile is available, you can deploy the EAR profile by selecting **Application** > **Deploy**. This option enables you to avoid using the SOA deployment wizard, if only task flow application deployment is necessary.
Overwrite Existing Application	Select to overwrite the existing version of the EAR file on the server.
Deployable Task Flow Projects	Select the task flow project WAR profiles to include in the EAR file. The task flow project WAR profiles are grouped in accordance with the composites that include the human task related to the task flow project. The context root of the WAR changes if the **Add generated profiles to application** checkbox is selected. **Note**: If you do not select a WAR profile, no task flows are deployed.

Table 40–7 (Cont.) Task Flow Deployment Dialog

Field	Description
■ **Projects**	Select from the list of deployable task flow projects or select the **Projects** checkbox to choose all available task flows. The task flows that display are based on the composites included in the SOA project or bundle selected for deployment.
■ **WAR Profiles**	Select the task flow project WAR files. Only the most recently created or modified task flow of the human task is available for selection.
■ **App Context Root**	Displays the application context root directory based on your selection for the WAR profile.

When you deploy a task form for a human task, as part of notification, the task form details are included in an email. For dynamic payloads, this email includes the content of the payload as it appears at that particular time.

For information about deploying SOA composite applications with task flows to multiple partition environments, see Section 40.7.1.2, "What You May Need to Know About Deploying Human Task Composites with Task Flows to Partitions."

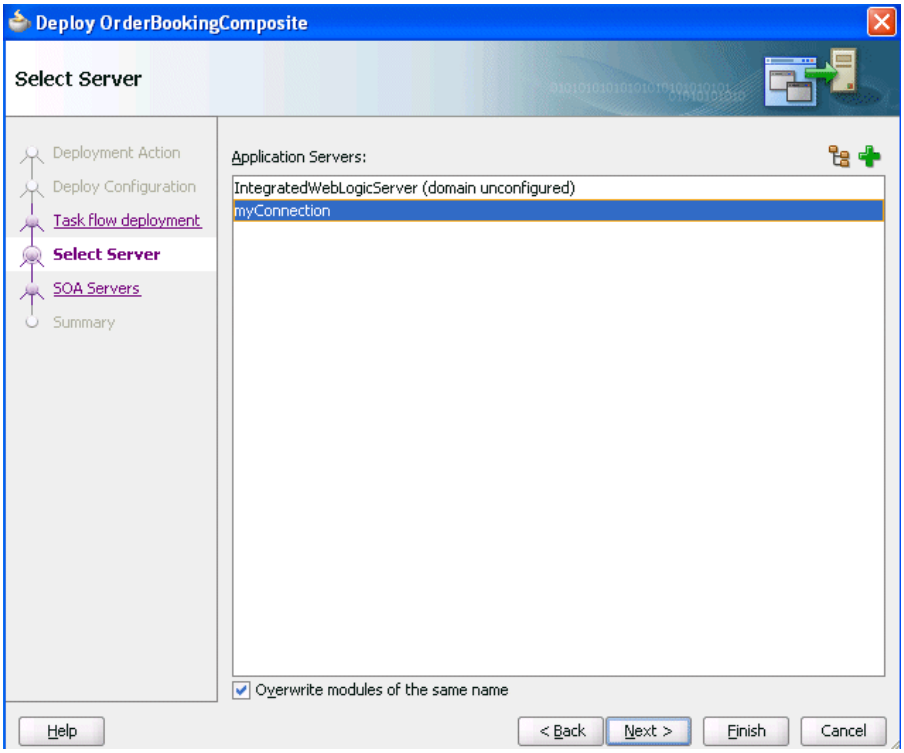
9. Click **Next**.

10. If you selected to deploy to an application server, the Select Server page appears for selecting an existing connection to an application server such as Oracle WebLogic Server from the list or clicking the **Add** icon to create a connection to a server. Figure 40–10 provides details.

    Otherwise, go to Step 15.

    Best Practice: It is recommended that task detail applications associated with a human workflow composite be deployed only to servers that have SOA configured on them, as well as the required ADF libraries.

Figure 40–10 Select Server Page

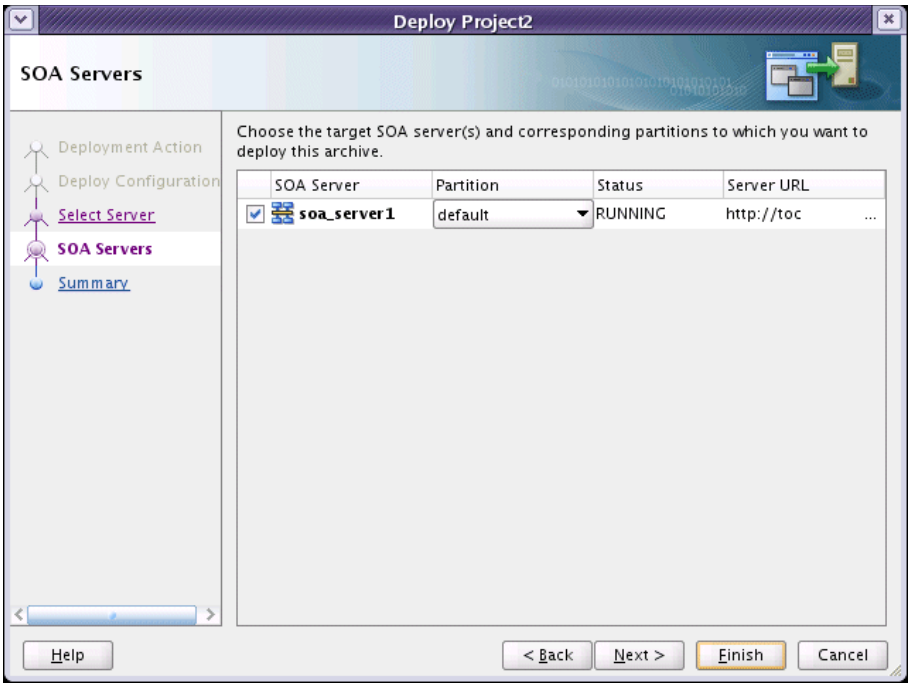


11. Click **Next**.
12. Select the target SOA servers to which to deploy this archive. If there are multiple servers or cluster nodes, select to deploy to one or more servers or nodes. Figure 40–11 provides details.
13. Select the partition in which to deploy this archive. If the server contains no partitions, you cannot deploy this archive. Also, if the server is not in a *running* state, you cannot deploy this archive. By default, a partition named **default** is automatically included with Oracle SOA Suite. You create partitions in the Manage Partitions page of Oracle Enterprise Manager Fusion Middleware Control.

Note: Human workflow artifacts such as task mapped attributes (previously known as flex field mappings) and rules (such as vacation rules) are defined based on the namespace of the task definition. Therefore, the following issues are true when the same SOA composite application with a human workflow task is deployed into multiple partitions:

- For the same task definition type, mapped attributes defined in one partition are visible in another partition.
- Rules defined on a task definition in one partition can apply to the same definition in another partition.

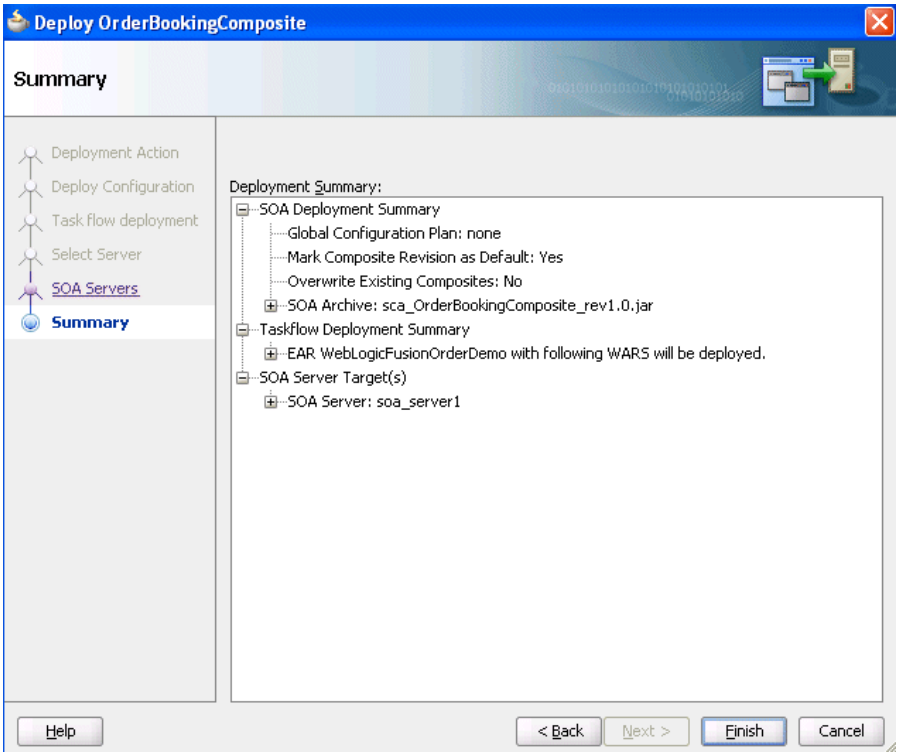
Figure 40–11 SOA Servers Page



14. Click **Next**.

15. Review the archive details on the Summary page shown in Figure 40–12, and click **Finish**.

Figure 40–12 Summary Page



16. If you selected to deploy to an application server, view the messages that display in the Deployment log window at the bottom of Oracle JDeveloper.
17. Enter the user name and password, and click **OK**.

    If deployment is successful, the following actions occur:

    - A JAR file for the SOA project is created under the **deploy** folder in Oracle JDeveloper with a naming convention of **sca**_*composite_name*_**rev***revision_number*.**jar**.
    - The project is displayed in the Resource Palette under *application_server_connection_name* > **SOA** > *SOA_server_name* > *partition_name*.
    - The project is displayed in the Application Server Navigator under *application_server_connection_name* > **SOA** > *SOA_server_name* > *partition_name*.

    You are now ready to monitor your application from Oracle Enterprise Manager Fusion Middleware Control. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for details.

    If deployment is unsuccessful, view the messages that display in the Deployment log window and take corrective actions. For more information, see Section 40.9, "Testing and Troubleshooting."

    For information on creating partitions, see the following documentation:

    - Section 40.7.5.2, "How to Manage SOA Composite Applications with ant Scripts"
    - *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*
    - *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*

    Note: If you want to redeploy the same version of a SOA composite application, you cannot change the composite name. You can deploy with the same revision number if you selected the **Overwrite any existing composites with the same revision ID** checkbox on the Deploy Configuration page.

40.7.1.2 What You May Need to Know About Deploying Human Task Composites with Task Flows to Partitions

To deploy a SOA composite application with a task flow from Oracle JDeveloper to a multiple partition environment, select the task flows to be deployed to the same partition in which the SOA composite application is being deployed.

When the task flow is deployed using only the EAR profile (deploying the task flow using the EAR deployer), the task flow is not partition-aware. Therefore, you must modify the `hwtaskflow.xml` file to include the partition name in the generated EAR file (the project version of the file remains unchanged). This file is located under the `TaskForm` project `adfmsrc` directory (for example, `HelpDeskRequestTaskFlow\adfmsrc\hwtaskflow.xml`). Example 40–5 provides details.

Example 40–5 hwtaskflow.xml file Modification

```
<hwTaskFlows
 xmlns="http://xmlns.oracle.com/bpel/workflow/hwTaskFlowProperties">
```

```
<ApplicationName>worklist</ApplicationName>
<LookupType>LOCAL</LookupType>
<TaskFlowDeploy>false</TaskFlowDeploy>
<PartitionName>partition2</PartitionName>
```

In addition, if you want to reuse the same task flow project for another partition, you must change the web context-root.

40.7.2 Deploying Multiple SOA Composite Applications in Oracle JDeveloper

You can deploy multiple SOA composite applications to an application server such as Oracle WebLogic Server at the same time by using the SOA bundle profile. This profile enables you to include one or more SAR profiles in the bundle and deploy the bundle to an application server.

> **Note:** You cannot deploy multiple SOA applications that are dependent upon one another in the same SOA bundle profile. For example, if application A calls application B, then you must first deploy application B separately.

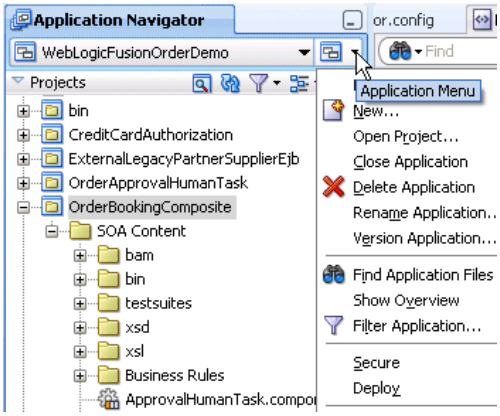
40.7.2.1 How to Deploy Multiple SOA Composite Applications

> **Note:** This section assumes you have created an application server connection. If not, see Section 40.7.1.1.1, "Creating an Application Server Connection" for instructions.

To deploy multiple SOA composite applications

1. From the **Application** menu, select **Application Properties**, as shown in Figure 40–13.

Figure 40–13 Application Properties

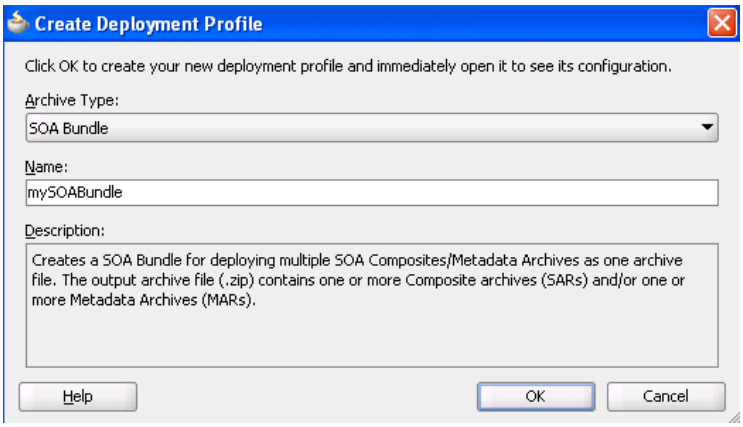


2. In the **Application Properties** dialog, click **Deployment**.
3. Click **New**.

   The Create Deployment Profile dialog appears.
4. In the **Archive Type** list, select **SOA Bundle**.
5. In the **Name** field, enter a name.

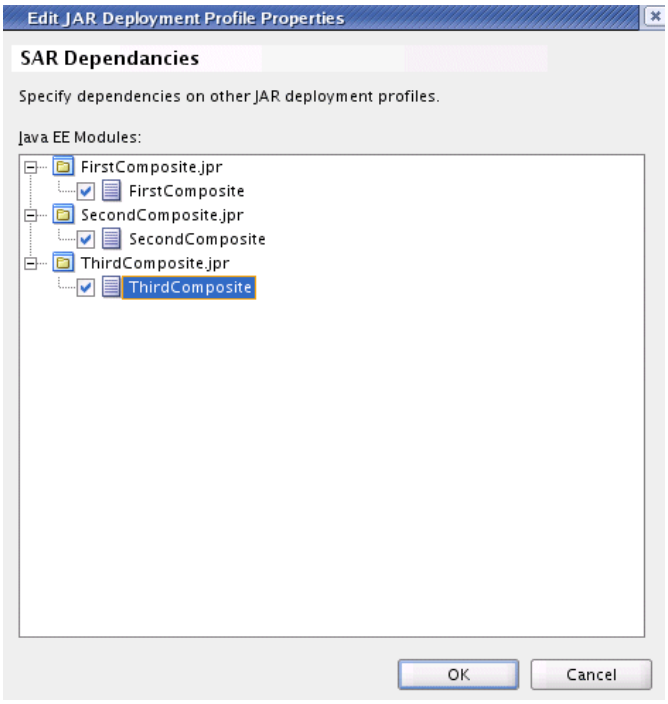
Figure 40–14 provides details.

Figure 40–14 Select the SOA Bundle



6. Click **OK**.
7. In the navigator on the left, select the **Dependencies** node.
8. Select the SARs you want to include in this bundle, as shown in Figure 40–15.

Figure 40–15 Select the SAR



9. Click **OK**.
10. Click **OK** to close the Application Properties dialog.
11. Select the **Application** menu again, then select **Deploy** > *SOA_Bundle_Name*.

    This invokes the deployment wizard.
12. See Step 3 on page 40-18 for details about responses to provide.

40.7.3 Deploying and Using Shared Metadata Across SOA Composite Applications in Oracle JDeveloper

This section describes how to deploy and use shared metadata across SOA composite applications.

40.7.3.1 How to Deploy Shared Metadata

Shared metadata is deployed to the SOA Infrastructure on the application server as a JAR file. The JAR file should contain all the resources to share. In Oracle JDeveloper, you can create a JAR profile for creating a shared artifacts archive.

All shared metadata is deployed to an existing SOA Infrastructure partition on the server. This metadata is deployed under the `/apps` namespace. For example, if you have a `MyProject/xsd/MySchema.xsd` file in the JAR file, then this file is deployed under the `/apps` namespace on the server. When you refer to this artifact in Oracle JDeveloper using a SOA-MDS connection, the URL becomes `oramds:/apps/MyProject/xsd/MySchema.xsd`.

This section describes how to perform the following tasks:

- Create a JAR profile and include the artifacts to share
- Create a SOA bundle that includes the JAR profile
- Deploy the SOA bundle to the application server

40.7.3.1.1 Create a JAR Profile and Include the Artifacts to Share

To create a JAR profile and include the artifacts to share:

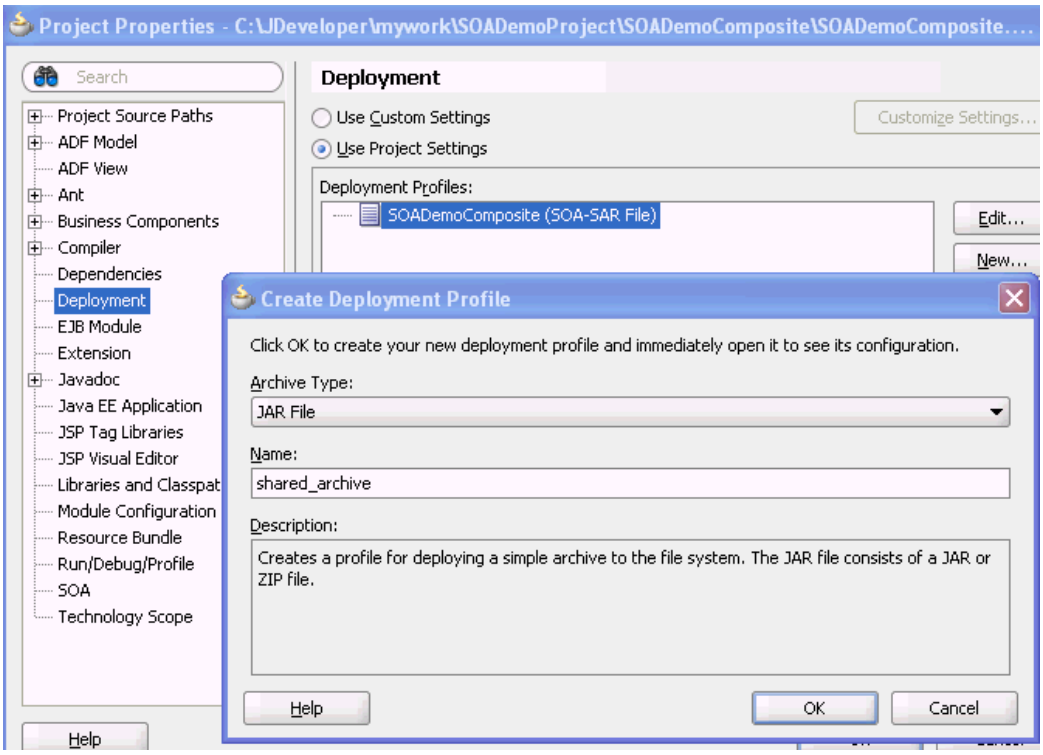
1. In the Application Navigator, right-click the SOA project.
2. Select **Project Properties**.

   The Project Properties dialog appears.
3. Click **Deployment** in the navigational tree on the left.
4. Click **New**.

   The Create Deployment Profile dialog appears.
5. From the **Archive Type** list, select **JAR File**.
6. In the **Name** field, enter a name (for this example, `shared_archive` is entered).

   The Create Deployment Profile dialog looks as shown in Figure 40–16.

Figure 40–16 JAR File Selection



7. Click **OK**.

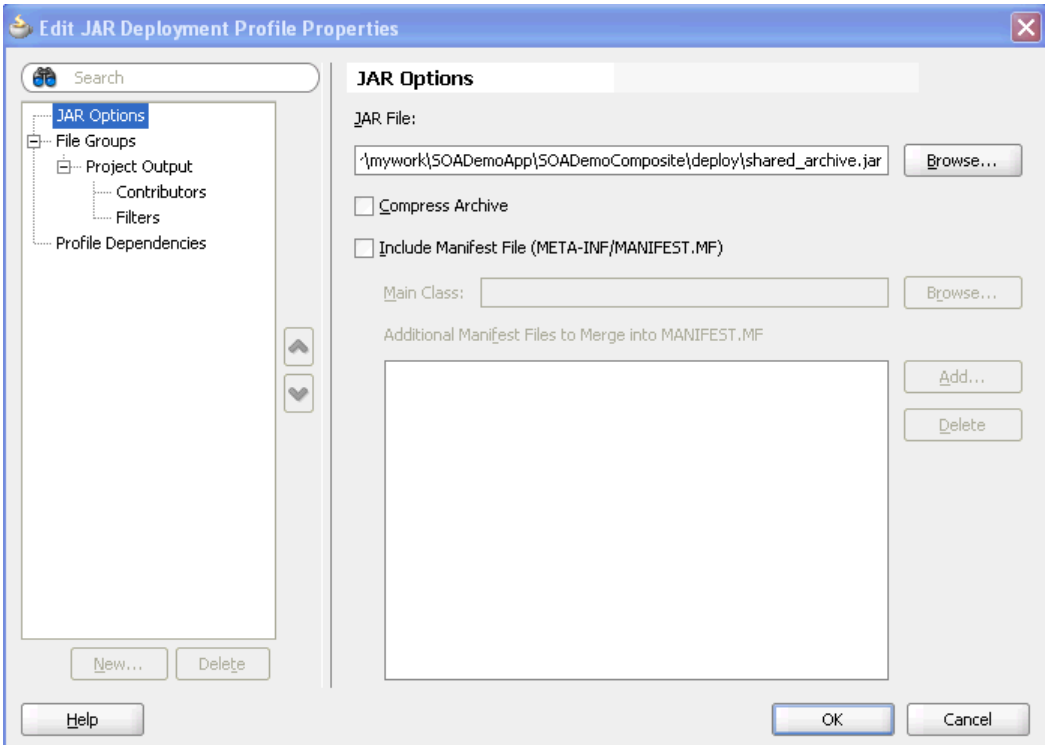
   The JAR Deployment Profile Properties dialog appears.

8. Select **JAR Options** from the navigational tree on the left.

9. Deselect **Include Manifest File (META-INF/MANIFEST.MF)**, as shown in Figure 40–17.

   This prevents the archive generator from adding the manifest file (`META-INF/MANIFEST.MF`) into the JAR file.

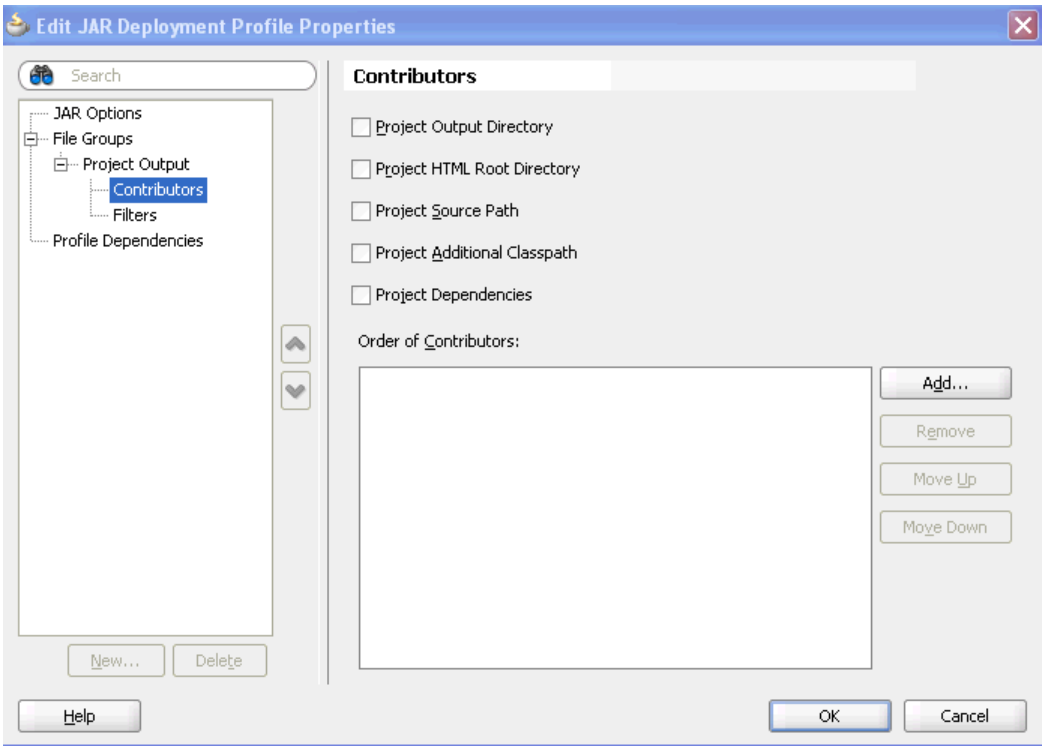
Figure 40–17 JAR File Options



10. Select **File Groups** > **Project Output** > **Contributors** from the navigational tree on the left.
11. Deselect the **Project Output Directory** and **Project Dependencies** options, as shown in Figure 40–18.

    This prevents the archive generator from adding the contents of the project output and project dependencies into the archive.

Figure 40–18 Contributors



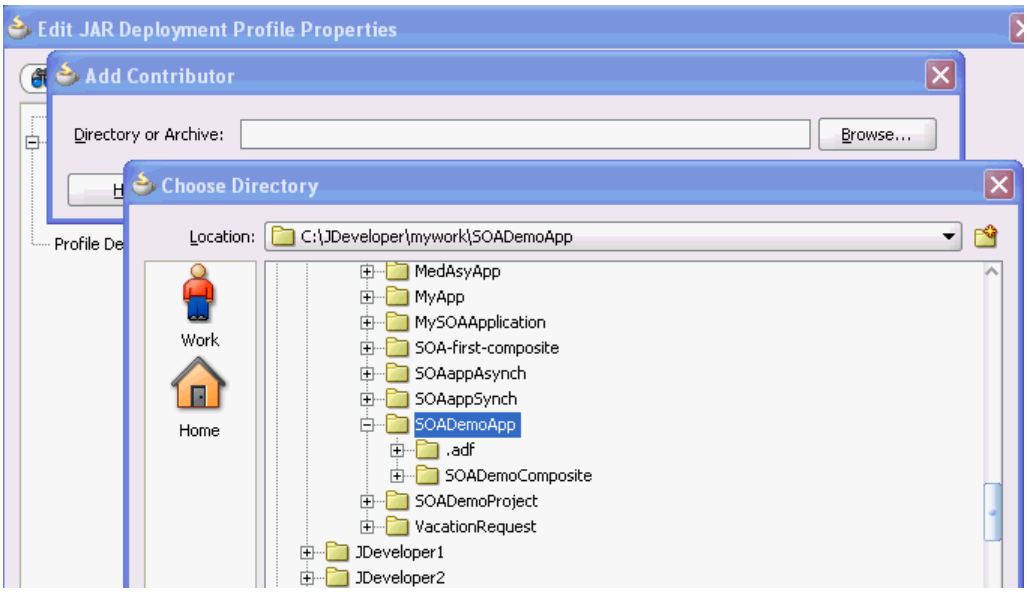
12. Click **Add** to add a new contributor.

    The Add Contributor dialog appears. This dialog enables you to add artifacts to your archive.

13. Click **Browse**.

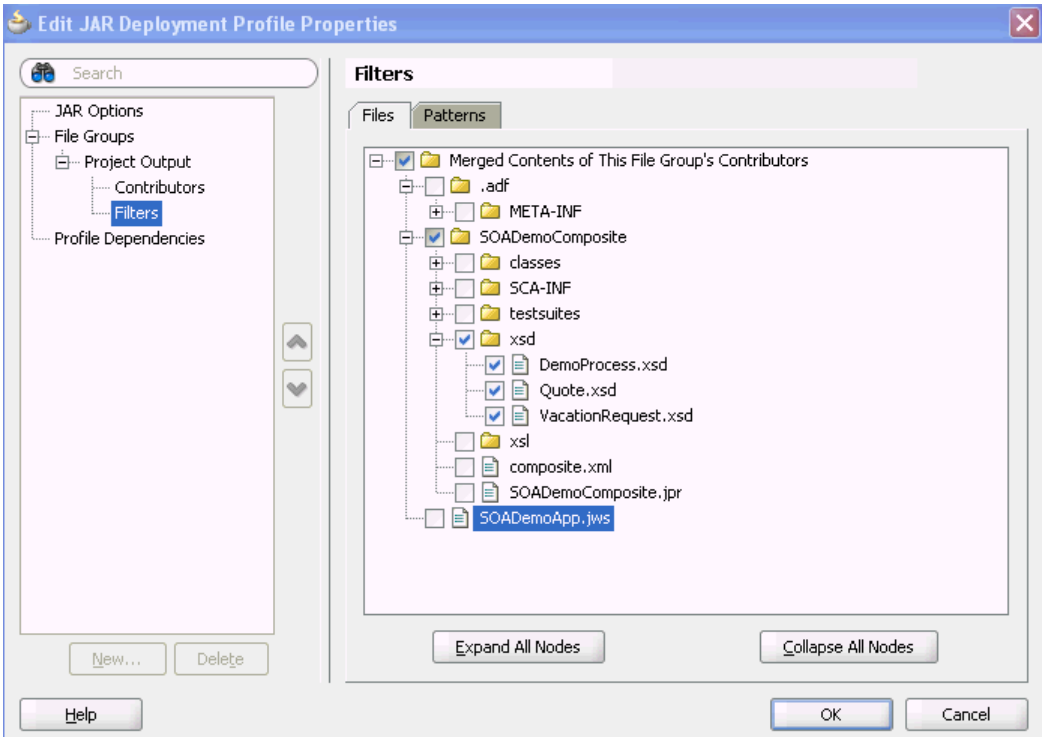
14. Select the folder in which your artifacts reside, as shown in Figure 40–19. Note that this also determines the hierarchy of artifacts in the archive.

Figure 40–19 Artifact Selection



15. Click **Select** to close the Choose Directory dialog.
16. Click **OK** to close the Add Contributor dialog.
17. Select **File Groups** > **Project Output** > **Filters** from the navigational tree on the left.
18. Select only the artifacts to include in the archive, as shown in Figure 40–20. For this example, the archive contains the following XSD files:
    - **SOADemoComposite/xsd/DemoProcess.xsd**
    - **SOADemoComposite/xsd/Quote.xsd**
    - **SOADemoComposite/xsd/VacationRequest.xsd**

Figure 40–20 Artifacts to Include in the Archive



19. Click **OK** to save changes to the JAR deployment profile.
20. Click **OK** to save the new deployment profile.
21. From the **File** main menu, select **Save All**.

40.7.3.1.2 Create a SOA Bundle that Includes the JAR Profile

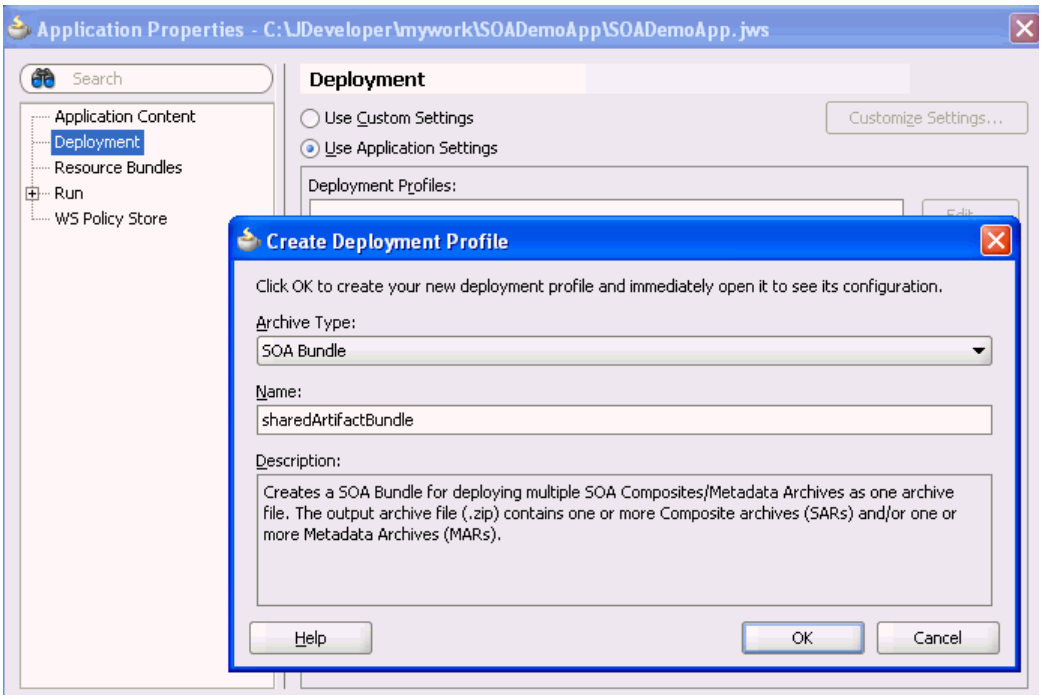
To create a SOA bundle that includes the JAR profile:

1. From the Application Menu, select **Application Properties** > **Deployment**.
2. Click **New** to create a SOA bundle profile.

   The Create Deployment Profile dialog appears.
3. From the **Archive Type** list, select **SOA Bundle**. A bundle is a collection of multiple SOA composite applications.

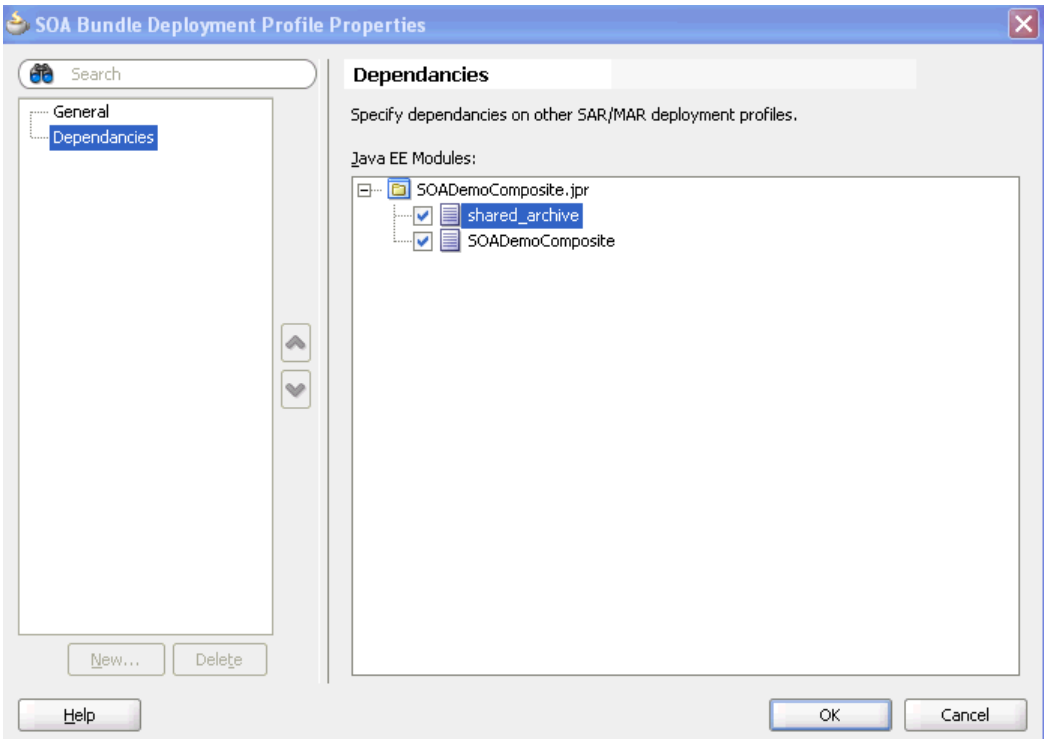
4. In the **Name** field, enter a name (for this example, `sharedArtifactBundle` is entered). Figure 40–21 provides details.

Figure 40–21 SOA Bundle Creation



5. Click **OK**.
6. Select **Dependencies** from the navigational tree on the left.
7. Select the JAR file and SOA-SAR profiles you previously created (for this example, named **shared_archive** and **sharedArtifactBundle**, respectively). You have the option of a JAR, a SOA-SAR, or both. Figure 40–22 provides details.

Figure 40–22 Deployment Profile Dependencies



8. Click **OK** to save the SOA bundle deployment profile changes.
9. Click **OK** to save the new deployment profile.
10. From the **File** main menu, select **Save All**.

40.7.3.1.3 Deploy the SOA Bundle

To deploy the SOA bundle:

1. Right-click the **Application** menu and select **Deploy** > *SOA_Bundle_Name*.

   This invokes the deployment wizard.

2. See Step 3 on page 40-18 for details about responses to provide.

   This deploys the SOA bundle to the application server (shared artifacts are deployed to the MDS database of Oracle SOA Suite).

40.7.3.2 How to Use Shared Metadata

This section describes how to browse and select the shared metadata you created in Section 40.7.3.1, "How to Deploy Shared Metadata."

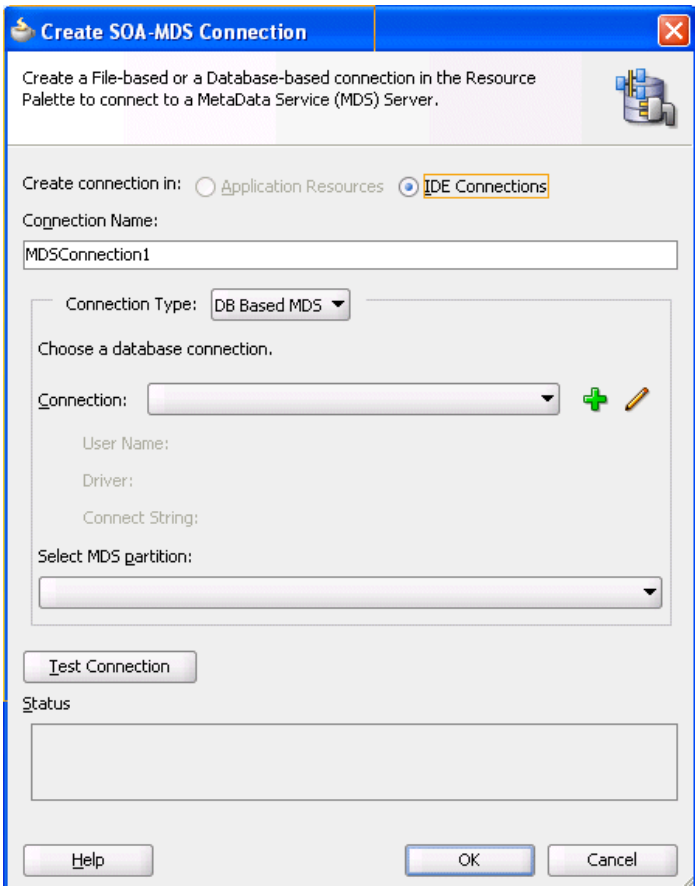
40.7.3.2.1 Creating a SOA-MDS Connection

To create a SOA-MDS connection:

1. From the **File** menu, select **New** > **Connections** > **SOA-MDS Connection**.

   The Create SOA-MDS Connection dialog shown in Figure 40–23 is displayed.

Figure 40–23 Create SOA-MDS Connection



2. Provide values appropriate to your environment, as shown in Table 40–8.

Table 40–8 Create SOA-MDS Connection Dialog

Field	Description
Create Connection In:	Ensure that **IDE Connection** is selected. This option enables the connection to display in the Resource Palette and be available to multiple applications. You cannot create a connection with the **Application Resources** option. This selection is disabled.
Connection Name	Enter a connection name. Upon successful completion of this connection creation, this name displays under **SOA-MDS** in the Resource Palette.
Connection Type	Select a connection type. An MDS repository can be file-based or database-based. The dialog is refreshed based on your selection. ▪ **DB based MDS** For most production environments, you use a database-based repository. Most components, such as Oracle SOA Suite, require that a schema be installed in a database, necessitating the use of a database-based repository. To use a database-based repository, you must first create it with the Repository Creation Utility. ▪ **File Based MDS**
Choose a database connection	Select an existing connection or create a new connection to the Oracle SOA Suite database with the MDS schema.

Table 40–8 (Cont.) Create SOA-MDS Connection Dialog

Field	Description
Select MDS Partition	Select the MDS partition (for example, **soa-infra**).
Test Connection	Click to test the SOA-MDS connection. **Note:** Even if the connection test fails, a connection is created.
Status	Displays status of the connection test.

3. Click **OK**.

   You can now browse the connection in the Resource Palette and view shared artifacts under the **/apps** node.

40.7.3.2.2 Creating a BPEL Process You can now browse and use the shared metadata from a different SOA composite application. For this example, you create a BPEL process service component in a different application.

To create a BPEL process:

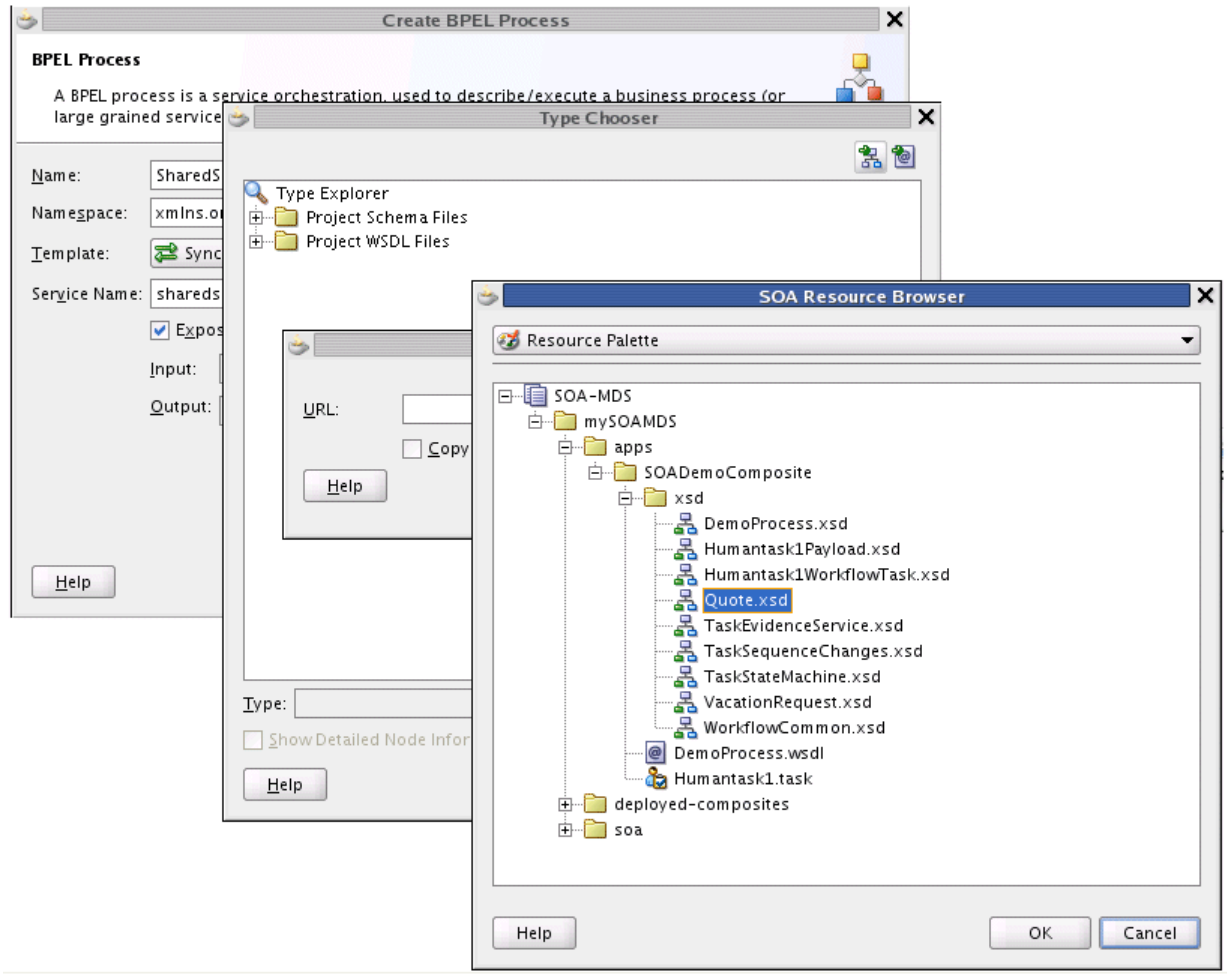
1. Create a new BPEL process service component in a different application.
2. In the Create BPEL Process dialog, click the **Browse** icon to the right of the **Input** field.

   The Type Chooser dialog appears.
3. In the upper right corner, click the **Import Schema File** icon.

   The Import Schema File dialog appears.
4. To the right of the **URL** field, click the **Browse** icon.

   The SOA Resource Browser dialog appears.
5. At the top of the dialog, select **Resource Palette** from the list.
6. Select shared metadata, as shown in Figure 40–24. For this example, the **Quote.xsd** file that you selected to include in the archive in Step 18 of Section 40.7.3.1.1, "Create a JAR Profile and Include the Artifacts to Share" is selected.

Figure 40–24 Shared Metadata in the SOA Resource Browser



7. Click **OK**.
8. In the Import Schema File dialog, click **OK**.
9. In the Type Chooser dialog, select a node of **Quote.xsd** (for this example, **QuoteRequest**), and click **OK**.
10. In the Create BPEL Process dialog, click **OK** to complete creation.
11. In the Application Navigator, select the WSDL file for the BPEL process.
12. Click **Source**.

    The WSDL file includes the following definition.

```
<wsdl:types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://www.mycompany.com/ns/salesquote"
 schemaLocation="oramds:/apps/SOADemoComposite/xsd/Quote.xsd" />
  </schema>
</wsdl:types>
```

13. Continue modeling the BPEL process as necessary.
14. Deploy the SOA composite application that includes the BPEL process.

40.7.4 Deploying an Existing SOA Archive in Oracle JDeveloper

You can deploy an existing SOA archive from the Application Server Navigator in Oracle JDeveloper.

Notes:

- The archive must exist. You cannot create an archive in the Deploy SOA Archive dialog.
- These instructions assume you have created an application server connection to an Oracle WebLogic Administration Server or another supported application server on which the SOA Infrastructure is deployed. Creating a connection to an Oracle WebLogic Administration Server enables you to browse for SOA composite applications deployed in the same domain. From the **File** main menu, select **New** > **Connections** > **Application Server Connection** to create a connection.

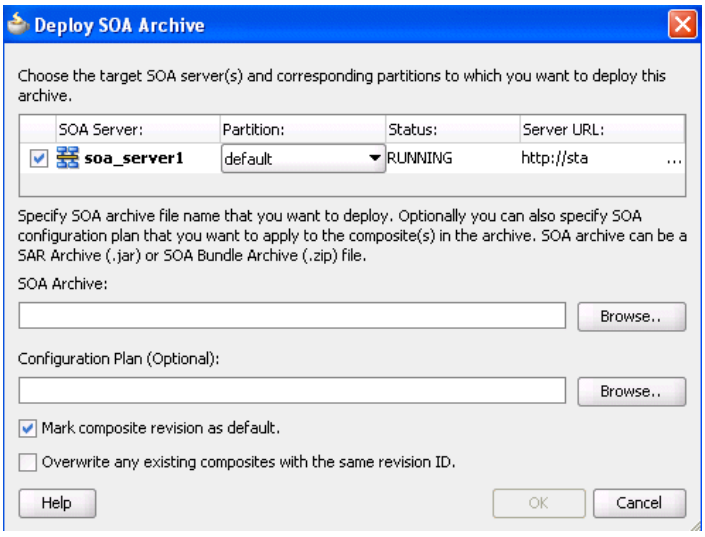
40.7.4.1 How to Deploy an Existing SOA Archive from Oracle JDeveloper

To deploy an existing SOA archive from Oracle JDeveloper:

1. From the **View** menu, select **Application Server Navigator**.
2. Expand your connection name.
3. Right-click the **SOA** folder.
4. Select **Deploy SOA Archive**.

   The Deploy SOA Archive dialog shown in Figure 40–25 appears.

Figure 40–25 Deploy SOA Archive Dialog



5. Provide responses appropriate to your environment, as described in Table 40–9.

Table 40–9 Create Deployment Profile Dialog Fields and Values

Field	Description
SOA Server	Select the SOA server to which to deploy the archive.
Partition	Select the partition in which to deploy the archive. If the server contains no partitions, you cannot deploy this archive. By default, a partition named **default** is automatically included with Oracle SOA Suite.
Status	Displays the status of the server. If the server is not in a running state, you cannot deploy this archive.
Server URL	Displays the URL of the server.
Choose target SOA server(s) to which you want to deploy this archive	Select the Oracle WebLogic Administration Server to which to deploy the archive.
SOA Archive	Click **Browse** to select a *prebuilt* SOA composite application archive. The archive consists of a JAR file of a single application or a SOA bundle ZIP file containing multiple applications.
Configuration Plan (Optional)	Click **Browse** to select a configuration plan to attach to the SOA composite application archive. The configuration plan enables you to define the URL and property values to use in different environments. During process deployment, the configuration plan is used to search the SOA project for values that must be replaced to adapt the project to the next target environment. For information about creating configuration plans, see Section 40.6.1.4, "How to Create a Configuration Plan in Oracle JDeveloper" or Section 40.6.1.5, "How to Create a Configuration Plan with the WLST Utility."
Mark composite revision as default	If you do not want the new revision to be the default, you can deselect this box. By default, a newly deployed composite revision is the default. This revision is instantiated when a new request comes in.
Overwrite any existing composites with the same revision ID	Select to overwrite (redeploy) an existing SOA composite application with the same revision ID. The consequences of this action are as follows: ■ A new version 1.0 of the SOA composite application is redeployed, overwriting a previously deployed 1.0 version. ■ The older, currently-deployed version of this revision is removed (overwritten). ■ If the older, currently-deployed version of this revision has running instances, the state of those instances is changed to stale.

6. Click **OK**.

For more information on deploying and testing SOA composite applications from the Application Server Navigator, see Section 2.8, "Managing and Testing a SOA Composite Application."

40.7.5 Managing SOA Composite Applications with Scripts

You can also manage SOA composite applications from a command line or scripting environment using the WLST scripting utility or `ant`. These options are well-suited for automation and can be easily integrated into existing release processes.

40.7.5.1 How to Manage SOA Composite Applications with the WLST Utility

You can manage SOA composite applications with the WLST scripting utility. For instructions, see *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.

40.7.5.2 How to Manage SOA Composite Applications with ant Scripts

You can manage SOA composite applications with the `ant` utility. `ant` is a Java-based build tool used by Oracle SOA Suite for managing SOA composite applications. The configuration files are XML-based and call out a target tree where various tasks are executed.

Table 40–10 lists the `ant` scripts available in the *`Middleware_Home`*`\`*`SOA_Suite_Home`*`\bin` directory.

Table 40–10 ant Management Scripts

Script	Description
`ant-sca-test.xml`	Automates the testing of SOA composite applications.
`ant-sca-compile.xml`	Compiles a SOA composite application.
`ant-sca-package.xml`	Packages a SOA composite application into a composite SAR file.
`ant-sca-deploy.xml`	Deploys a SOA composite application.
`ant-sca-deploy.xml undeploy`	Undeploys a SOA composite application.
`ant-sca-deploy.xml exportComposite`	Exports a composite into a SAR file.
`ant-sca-deploy.xml exportUpdates`	Exports postdeployment changes of a composite into a JAR file.
`ant-sca-deploy.xml importUpdates`	Imports postdeployment changes of a composite.
`ant-sca-deploy.xml exportSharedData`	Exports shared data of a given pattern into a JAR file.
`ant-sca-deploy.xml removeSharedData`	Removes a top-level shared data folder.
`ant-sca-mgmt.xml startComposite`	Starts a SOA composite application.
`ant-sca-mgmt.xml stopComposite`	Stops a SOA composite application.
`ant-sca-mgmt.xml activateComposite`	Activates a SOA composite application.
`ant-sca-mgmt.xml retireComposite`	Retires a SOA composite application.
`ant-sca-mgmt.xml assignDefaultComposite`	Assigns a default revision version.
`ant-sca-mgmt.xml listDeployedComposites`	Lists deployed SOA composite applications.
`ant-sca-mgmt.xml listPartitions`	Lists all available partitions in the SOA Infrastructure.

Table 40–10 (Cont.) ant Management Scripts

Script	Description
`ant-sca-mgmt.xml listCompositesInPartition`	Lists all composites in a partition.
`ant-sca-mgmt.xml createPartition`	Creates a partition in the SOA Infrastructure.
`ant-sca-mgmt.xml deletePartition`	Undeploys all composites in a partition before deleting the partition.
`ant-sca-mgmt.xml startCompositesInPartition`	Starts all composites in a partition.
`ant-sca-mgmt.xml stopCompositesInPartition`	Stops all composites in a partition.
`ant-sca-mgmt.xml activateCompositesInPartition`	Activates all composites in a partition.
`ant-sca-mgmt.xml retireCompositesInPartition`	Retires all composites in a partition.
`ant-sca-upgrade.xml`	Migrates BPEL and ESB release 10.1.3 metadata to release 11*g*. **Note:** If any Java code is part of the project, you must manually modify the code to pass compilation with an 11*g* compiler. For BPEL process instance data, active data used by the 10.1.3 Oracle BPEL Server is not migrated.

For additional information about `ant`, visit the following URL:

`http://ant.apache.org`

40.7.5.2.1 Testing a SOA Composite Application

Example 40–6 provides an example of executing a test case. Test cases enable you to automate the testing of SOA composite applications.

Example 40–6 Testing an Application

```
ant -f ant-sca-test.xml -Dscatest.input=MyComposite
-Djndi.properties=/home/jdoe/jndi.properties
```

Table 40–11 describes the syntax.

Table 40–11 ant Testing Commands

Argument	Definition
scatest	Possible inputs are as follows: ■ java.passed.home The script picks this from the environment value of JAVA_HOME. Provide this input to override. ■ wl_home This is the location of Oracle WebLogic Server home (defaults to *Oracle_Home*/.../wlserver_10.3). ■ scatest.input The name of the composite to test. ■ scatest.format The format of the output file (defaults to native; the other option is junit). ■ scatest.result The result directory in which to place the output files (defaults to *temp_dir*/out). ■ jndi.properties.input The jndi.properties file to use.
jndi. properties	Absolute path to the JNDI property file. This is a property file that contains JNDI properties for connecting to the server. For example: java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory java.naming.provider.url=t3://myserver.us.oracle.com:8001/soa-infra java.naming.security.principal=weblogic dedicated.connection=true dedicated.rmicontext=true Since a composite test (in a test suite) is executed on the SOA Infrastructure, this properties file contains the connection information. For this example, these properties create a connection to the SOA Infrastructure hosted in myserver.us.oracle.com, port 8001 and use a user name of weblogic. You are prompted to specify the password. You typically create one jndi.properties file (for example, in /home/myhome/jndi.properties) and use it for all test runs.

For more information on creating and running tests on SOA composite applications, see Chapter 41, "Automating Testing of SOA Composite Applications" and *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

40.7.5.2.2 Compiling a SOA Composite Application Example 40–7 provides an example of compiling a SOA composite application, which validates it for structure and syntax.

Example 40–7 Compiling an Application

```
ant -f ant-sca-compile.xml
-Dscac.input=/myApplication/myComposite/composite.xml
```

Table 40–12 describes the syntax.

Table 40–12 ant Compiling Commands

Argument	Definition
scac	Possible inputs are as follows: ■ java.passed.home The script picks this from the environment value of JAVA_HOME. Provide this input to override. ■ wl_home This is the location of Oracle WebLogic Server home (defaults to *Oracle_Home*/.../wlserver_10.3). ■ scac.input The composite.xml file to compile. ■ scac.output The output file with scac results (defaults to *temp_dir*/out.xml). ■ scac.error The file with scac errors (defaults to *temp_dir*/out.err). ■ scac.application.home The application home directory of the composite being compiled. ■ scac.displayLevel Controls the level of logs written to scac.output file. The value can be 1, 2, or 3 (this defaults to 1).

40.7.5.2.3 Packaging a SOA Composite Application into a Composite SAR File

Example 40–8 provides an example of packaging a SOA composite application into a composite SAR file. The outcome of this command is a SOA archive. Check the output of the command for the exact location of the resulting file.

Example 40–8 Packaging an Application

```
ant -f ant-sca-package.xml
-DcompositeDir=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing\POProcessing
-DcompositeName=POProcessing
-Drevision=6-cmdline
-Dsca.application.home=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing
```

Table 40–13 describes the syntax.

Table 40–13 ant Packaging Commands

Argument	Definition
compositeDir	Absolute path of a directory that contains composite artifacts.
compositeName	Name of the composite.
revision	Revision ID of the composite.
sca.application.home	Optional. Absolute path of the application home directory. This property is required if you have shared data.
oracle.home	Optional. The oracle.home property.

40.7.5.2.4 Deploying a SOA Composite Application Example 40–9 provides an example of deploying a SOA composite application.

Example 40–9 Deploying an Application

```
ant -f ant-sca-deploy.xml
-DserverURL=http://localhost:8001
-DsarLocation=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing\POPro
cessing\deploy\sca_POProcessing_rev6-cmdline.jar
-Doverwrite=true
-Duser=weblogic
-DforceDefault=true
-Dconfigplan=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing\POProc
 essing\demed_cfgplan.xml
-Dpartition=partition.name
```

Note: After specifying the user name, enter the password when prompted.

Table 40–14 describes the syntax.

Table 40–14 ant Deployment Commands

Argument	Definition
`serverURL`	URL of the server that hosts the SOA Infrastructure application (for example, `http://myhost10:8001`).
`sarLocation`	Absolute path to one the following: ■ SAR file. ■ ZIP file that includes multiple SARs.
`overwrite`	Optional. Indicates whether to overwrite an existing SOA composite application on the server. ■ `false` (default): Does not overwrite the file. ■ `true`: Overwrites the file.
`user`	Optional. User name to access the composite deployer servlet when basic authentication is configured.
`password`	Optional. Password to access the composite deployer servlet when basic authentication is configured. If you enter the user name, you are prompted to enter the password if you do not provide it here.
`forceDefault`	Optional. Indicates whether to set the version being deployed as the default version for that composite application. ■ `true` (default): Makes it the default composite. ■ `false`: Does not make it the default composite.
`configplan`	Absolute path of a configuration plan to be applied to a specified SAR file or to all SAR files included in the ZIP file.
`sysPropFile`	Passes in a system properties file that is useful for setting extra system properties, for debugging, for SSL configuration, and so on. If you specify a file name (for example, `tmp-sys.properties`), you can define properties such as the following: `javax.net.debug=all`

Table 40–14 (Cont.) ant Deployment Commands

Argument	Definition
`partition`	Optional. The name of the partition in which to deploy the SOA composite application. The default value is `default`. If you do not specify a partition, the composite is automatically deployed into the `default` partition.

Note: Human workflow artifacts such as task mapped attributes (previously known as flex field mappings) and rules (such as vacation rules) are defined based on the namespace of the task definition. Therefore, the following issues are true when the same SOA composite application with a human workflow task is deployed into multiple partitions:

- For the same task definition type, mapped attributes defined in one partition are visible in another partition.
- Rules defined on a task definition in one partition can apply to the same definition in another partition.

40.7.5.2.5 Undeploying a SOA Composite Application Example 40–10 provides an example of undeploying a SOA composite application.

Example 40–10 Undeploying a SOA Composite Application

```
ant -f ant-sca-deploy.xml undeploy
-DserverURL=http://localhost:8001
-DcompositeName=POProcessing
-Drevision=rev6-cmdline
-Duser=weblogic
-Dpartition=partition.name
```

Note: After specifying the user name, enter the password when prompted.

Table 40–15 describes the syntax.

Table 40–15 ant Undeployment Commands

Argument	Definition
`serverURL`	URL of the server that hosts the SOA Infrastructure application (for example, `http://myhost10:7001`).
`compositeName`	Name of the SOA composite application.
`revision`	Revision ID of the SOA composite application.
`user`	Optional. User name to access the composite deployer servlet when basic authentication is configured. If you enter the user name, you are prompted to enter the corresponding password.
`password`	Optional. Password to access the composite deployer servlet when basic authentication is configured.

Table 40–15 (Cont.) ant Undeployment Commands

Argument	Definition
`partition`	Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched.

40.7.5.2.6 Exporting a Composite into a SAR File Example 40–11 provides an example of exporting a composite into a SAR file.

Example 40–11 Exporting a Composite into a SAR File

```
ant -f ant-sca-deploy.xml exportComposite -DserverURL=server.url
 -DupdateType=update.type -DsarFile=sar.file
 -DcompositeName=composite.name -Drevision=revision -Duser=user
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–16 describes the syntax.

Table 40–16 ant Export Commands

Argument	Definition
`serverURL`	The URL of the server that hosts the SOA Infrastructure application (for example, `http://stabc:8001`).
`updateType`	The type of postdeployment changes to be included: ▪ `none`: No postdeployment changes are included. ▪ `all`: All postdeployment changes are included. ▪ `property`: Property changes are included (binding component properties, composite properties such as audit level settings and payload validation status, and policy attachments). ▪ `runtime`: Postdeployment runtime changes are included (rules dictionary and domain value maps (DVMs)).
`sarFile`	The absolute path of the SAR file to be generated.
`compositeName`	The name of the composite to be exported.
`revision`	The revision of the composite to be exported.
`user`	Optional. The user name for accessing the server when basic configuration is configured.
`password`	Optional. The password for accessing the server when basic configuration is configured.

Example 40–12 shows how to export a composite without including any postdeployment changes.

Example 40–12 Exporting a Composite Without Including Any Postdeployment Changes

```
ant -f ant-sca-deploy.xml exportComposite -DserverURL=http://stabc:8001
 -DupdateType=none
 -DsarFile=/tmp/sca_HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

Example 40–13 shows how to export a composite with all postdeployment changes.

Example 40–13 Exporting a Composite With All Postdeployment Changes

```
ant -f ant-sca-deploy.xml exportComposite -DserverURL=http://stabc:8001
 -DupdateType=all
 -DsarFile=/tmp/sca_HelloWorld_rev1.0-all.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

Example 40–14 shows how to export a composite with property postdeployment updates.

Example 40–14 Exporting a Composite With Property Postdeployment Updates

```
ant -f ant-sca-deploy.xml exportComposite -DserverURL=http://stabc:8001
 -DupdateType=property
 -DsarFile=/tmp/sca_HelloWorld_rev1.0-prop.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

Example 40–15 shows how to export a composite with runtime/metadata postdeployment updates.

Example 40–15 Exporting a Composite With Runtime/Metadata Postdeployment Updates

```
ant -f ant-sca-deploy.xml exportComposite -DserverURL=http://stabc:8001
 -DupdateType=runtime
 -DsarFile=/tmp/sca_HelloWorld_rev1.0-runtime.jar
 -DcompositeName=HelloWorld -Drevision=1.0
```

40.7.5.2.7 Exporting Postdeployment Changes of a Composite into a JAR File Example 40–16 provides an example of exporting postdeployment changes of a composite into a JAR file.

Example 40–16 Exporting Postdeployment Changes of a Composite into a JAR File

```
ant -f ant-sca-deploy.xml exportUpdates -DserverURL=server.url
 -DupdateType=update.type -DjarFile=jar.file
 -DcompositeName=composite.name -Drevision=revision -Duser=user
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–17 describes the syntax.

Table 40–17 ant Postdeployment Export Commands

Argument	Definition
`serverURL`	The URL of the server that hosts the SOA Infrastructure application (for example, `http://stabc:8001`).
`updateType`	The type of postdeployment changes to be exported. ▪ `all`: Includes all postdeployment changes. ▪ `property`: Includes only property postdeployment changes (binding component properties, composite properties such as audit level settings and payload validation status, and policy attachments). ▪ `runtime`: Includes only runtime (rules dictionary and domain value maps (DVMs)).

Table 40–17 (Cont.) ant Postdeployment Export Commands

Argument	Definition
`jarFile`	The absolute path of the JAR file to be generated.
`compositeName`	The name of the composite to be exported.
`revision`	The revision of the composite to be exported.
`user`	Optional. The user name for accessing the server when basic configuration is configured.
`password`	Optional. The password for accessing the server when basic configuration is configured.

Example 40–17 shows how to export all postdeployment updates.

Example 40–17 Exporting All Postdeployment Updates

```
ant -f ant-sca-deploy.xml exportUpdates -DserverURL=http://stabc:8001
 -DupdateType=all
 -DjarFile=/tmp/all-HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

Example 40–18 shows how to export property postdeployment updates.

Example 40–18 Exporting Property Postdeployment Updates

```
ant -f ant-sca-deploy.xml exportUpdates -DserverURL=http://stabc:8001
 -DupdateType=property
 -DjarFile=/tmp/prop-HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

Example 40–19 shows how to export runtime/metadata postdeployment updates.

Example 40–19 Exporting Runtime/Metadata Postdeployment Updates

```
ant -f ant-sca-deploy.xml exportUpdates -DserverURL=http://stabc:8001
 -DupdateType=runtime
 -DjarFile=/tmp/runtime-HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

40.7.5.2.8 Importing Postdeployment Changes of a Composite Example 40–20 provides an example of importing postdeployment changes of a composite.

Example 40–20 Importing Postdeployment Changes of a Composite

```
ant -f ant-sca-deploy.xml importUpdates -DserverURL=server.url -DjarFile=jar.file
 -DcompositeName=composite.name -Drevision=revision -Duser=user
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–18 describes the syntax.

Table 40–18 ant Postdeployment Import Commands

Argument	Definition
`serverURL`	The URL of the server that hosts the SOA Infrastructure application (for example, `http://stabc:8001`).
`jarFile`	The absolute path of the JAR file that contains postdeployment changes.
`compositeName`	The name of the composite into which the postdeployment changes are imported.
`revision`	The revision of the composite to which the postdeployment changes are imported.
`user`	Optional. The user name for accessing the server when basic configuration is configured.
`password`	Optional. The password for accessing the server when basic configuration is configured.

Example 40–21 shows how to import postdeployment changes of a composite.

Example 40–21 Importing Postdeployment Changes of a Composite

```
ant -f ant-sca-deploy.xml importUpdates -DserverURL=http://stabc:8001
 -DjarFile=/tmp/prop-HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

40.7.5.2.9 Exporting Shared Data of a Given Pattern into a JAR File Example 40–22 provides an example of exporting shared data of a given pattern into a JAR file.

Example 40–22 Exporting Shared Data of a Given Pattern into a JAR File

```
ant -f ant-sca-deploy.xml exportSharedData -DserverURL=server.url
 -DjarFile=jar.file -Dpattern=pattern -Duser=user
```

Note: After specifying the user name, enter the password when prompted.

Table 40–19 describes the syntax.

Table 40–19 ant Shared Data Export Commands

Argument	Definition
`serverURL`	The URL of the server that hosts the SOA Infrastructure application (for example, `http://stabc:8001`).
`jarFile`	The absolute path of the JAR file to be generated.
`pattern`	The file pattern supported by MDS transfer APIs. Use the semicolon delimiter (`;`) if multiple patterns are specified. Exclude the shared data namespace `/apps` in the pattern. For example: `/Project1/**;/Project2/**` This example exports all documents under `/apps/Project1` and `/apps/Project2`.
`user`	Optional. The user name for accessing the server when basic configuration is configured.
`password`	The password for accessing the server when basic configuration is configured. This parameter is optional.

Example 40–23 shows how to export shared data of a given pattern into a JAR file.

Example 40–23 Exporting Shared Data of a Given Pattern into a JAR File

```
ant -f ant-sca-deploy.xml exportSharedData -DserverURL=http://stabc:8001
 -DjarFile=/tmp/MySharedData.jar
 -Dpattern="/Project1/**"
```

40.7.5.2.10 Removing a Top-level Shared Data Folder Example 40–24 provides an example of removing a top-level shared data folder, even if there are composites deployed in the service engine.

Example 40–24 Removing a Top-level Shared Data Folder

```
ant -f ant-sca-deploy.xml removeSharedData -DserverURL=server.url
 -DfolderName=folder.name -Duser=user
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–20 describes the syntax.

Table 40–20 ant Shared Data Folder Removal Commands

Argument	Definition
`serverURL`	URL of the server that hosts the SOA Infrastructure application (for example, `http://myhost10:8001`).
`foldername`	The name of the top-level shared data folder to remove.
`user`	Optional. The user name for accessing the server when basic configuration is configured.
`password`	Optional. The password for accessing the server when basic configuration is configured.

Example 40–25 shows how to remove a top-level shared data folder named `Project1`.

Example 40–25 Removing a Top-level Shared Data Folder

```
ant -f ant-sca-deploy.xml removeSharedData -DserverURL=http://stabc:8001
 -DfolderName=Project1
```

40.7.5.2.11 Starting a SOA Composite Application Example 40–26 provides an example of starting a SOA composite application.

Example 40–26 Starting an Application

```
ant -f ant-sca-mgmt.xml startComposite -Dhost=myhost -Dport=8001 -Duser=weblogic
  -DcompositeName=HelloWorld -Drevision=1.0 -Dpartition=partition.name
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–21 describes the syntax.

Table 40–21 ant SOA Composite Application Startup Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`compositeName`	Name of the SOA composite application.
`revision`	Revision of the SOA composite application.
`label`	Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one.
`partition`	Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched.

40.7.5.2.12 Stopping a SOA Composite Application Example 40–27 provides an example of stopping a SOA composite application.

Example 40–27 Stopping an Application

```
ant -f ant-sca-mgmt.xml stopComposite -Dhost=myhost -Dport=8001 -Duser=weblogic
-DcompositeName=HelloWorld -Drevision=1.0 -Dpartition=partition.name
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–22 describes the syntax.

Table 40–22 ant SOA Composite Application Stop Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`compositeName`	Name of the SOA composite application.
`revision`	Revision of the SOA composite application.
`label`	Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one.
`partition`	Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched.

40.7.5.2.13 Activating a SOA Composite Application Example 40–28 provides an example of activating a SOA composite application.

Example 40–28 Activating an Application

```
ant -f ant-sca-mgmt.xml activateComposite -Dhost=myhost -Dport=8001
-Duser=weblogic-DcompositeName=HelloWorld -Drevision=1.0
-Dpartition=partition.name
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–23 describes the syntax.

Table 40–23 ant SOA Composite Application Activation Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`compositeName`	Name of the SOA composite application.
`revision`	Revision of the SOA composite application.
`label`	Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one.
`partition`	Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched.

40.7.5.2.14 Retiring a SOA Composite Application Example 40–29 provides an example of retiring a SOA composite application.

Example 40–29 Retiring an Application

```
ant -f ant-sca-mgmt.xml retireComposite -Dhost=myhost -Dport=8001 -Duser=weblogic
-DcompositeName=HelloWorld -Drevision=1.0 -Dpartition=partition.name
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–24 describes the syntax.

Table 40–24 ant SOA Composite Application Retirement Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).

Table 40–24 (Cont.) ant SOA Composite Application Retirement Commands

Argument	Definition
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`compositeName`	Name of the SOA composite application.
`revision`	Revision of the SOA composite application.
`label`	Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one.
`partition`	Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched.

40.7.5.2.15 Assigning the Default Version to a SOA Composite Application

Example 40–30 provides an example of assigning the default version to a SOA composite application.

Example 40–30 Assigning the Default Version to a SOA Composite Application

```
ant -f ant-sca-mgmt.xml assignDefaultComposite -Dhost=myhost -Dport=8001
-Duser=weblogic -DcompositeName=HelloWorld -Drevision=1.0
-Dpartition=partition.name
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–25 describes the syntax.

Table 40–25 ant SOA Composite Application Default Version Assignment Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`compositeName`	Name of the SOA composite application.
`revision`	Revision of the SOA composite application.
`partition`	Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched.

40.7.5.2.16 Listing the Deployed SOA Composite Applications

Example 40–31 provides an example of listing the deployed SOA composite applications.

Example 40–31 Listing the Deployed SOA Composite Applications

```
ant -f ant-sca-mgmt.xml listDeployedComposites -Dhost=myhost -Dport=8001
-Duser=weblogic
```

Note: After specifying the user name, enter the password when prompted.

Table 40–26 describes the syntax.

Table 40–26 ant SOA Composite Application Deployment List Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.

40.7.5.2.17 Listing All Available Partitions in the SOA Infrastructure Example 40–32 provides the syntax for listing all available partitions in the SOA Infrastructure.

Example 40–32 Listing All Available Partitions in the SOA Infrastructure

```
ant -f ant-sca-mgmt.xml listPartitions -Dhost=host -Dport=port -Duser=user
```

Note: After specifying the user name, enter the password when prompted.

Table 40–27 describes the syntax.

Table 40–27 ant SOA Infrastructure Partitioning List Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.

Example 40–33 provides an example of listing all available partitions in the SOA Infrastructure.

Example 40–33 Listing All Available Partitions in the SOA Infrastructure

```
ant -f ant-sca-mgmt.xml listPartitions -Dhost=stabc10 -Dport=8001
```

40.7.5.2.18 Listing All Composites in a Partition Example 40–34 provides the syntax for listing all composites in a partition.

Example 40–34 Listing All Composites in a Partition

```
ant -f ant-sca-mgmt.xml listCompositesInPartition -Dhost=host -Dport=port
-Duser=user -Dpartition=partition.name
```

Note: After specifying the user name, enter the password when prompted.

Table 40–28 describes the syntax.

Table 40–28 ant Composite Partitioning List Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`partition`	The name of the partition.

Example 40–35 provides an example of listing all composites in a partition named `myPartition`.

Example 40–35 Listing All Composites in a Partition

```
ant -f ant-sca-mgmt.xml listCompositesInPartition -Dhost=stabc10 -Dport=8001
-Dpartition=myPartition
```

40.7.5.2.19 Creating a Partition in the SOA Infrastructure Example 40–36 provides the syntax for creating a partition in the SOA Infrastructure.

Example 40–36 Creating a Partition in the SOA Infrastructure

```
ant -f ant-sca-mgmt.xml createPartition -Dhost=host -Dport=port -Duser=user
-Dpartition=partition.name
```

Note: After specifying the user name, enter the password when prompted.

Table 40–29 describes the syntax.

Table 40–29 ant Partition Creation Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`partition`	The name of the partition to create.

Example 40–37 provides an example of creating a partition in the SOA Infrastructure named `myPartition`.

Example 40–37 Creating a Partition in the SOA Infrastructure

```
ant -f ant-sca-mgmt.xml createPartition -Dhost=stabc10 -Dport=8001
-Dpartition=myPartition
```

40.7.5.2.20 Deleting a Partition in the SOA Infrastructure Example 40–38 provides the syntax for deleting a partition in the SOA Infrastructure. This command undeploys all composites in the partition before deleting the partition.

Example 40–38 Deleting a Partition in the SOA Infrastructure

```
ant -f ant-sca-mgmt.xml deletePartition -Dhost=host -Dport=port -Duser=user
-Dpartition=partition.name
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–30 describes the syntax.

Table 40–30 ant Partition Deletion Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`partition`	The name of the partition to delete.

Example 40–39 provides an example of deleting a partition in the SOA Infrastructure named `myPartition`.

Example 40–39 Deleting a Partition in the SOA Infrastructure

```
ant -f ant-sca-mgmt.xml deletePartition -Dhost=stabc10 -Dport=8001
-Dpartition=myPartition
```

40.7.5.2.21 Starting All Composites in the Partition Example 40–40 provides the syntax for starting all composites in the partition.

Example 40–40 Starting All Composites in the Partition

```
ant -f ant-sca-mgmt.xml startCompositesInPartition -Dhost=host -Dport=port
-Duser=user -Dpartition=partition.name
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–31 describes the syntax.

Table 40–31 ant Partition Startup Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).

Table 40–31 (Cont.) ant Partition Startup Commands

Argument	Definition
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`partition`	The name of the partition.

Example 40–41 provides an example of starting all composites in the partition named `myPartition`.

Example 40–41 Starting All Composites in the Partition

```
ant -f ant-sca-mgmt.xml startCompositesInPartition -Dhost=stabc10 -Dport=8001
-Dpartition=myPartition
```

40.7.5.2.22 Stopping All Composites in the Partition Example 40–42 provides the syntax for stopping all composites in the partition.

Example 40–42 Stopping All Composites in the Partition

```
ant -f ant-sca-mgmt.xml stopCompositesInPartition -Dhost=host -Dport=port
-Duser=user -Dpartition=partition.name
```

Note: After specifying the user name, enter the password when prompted.

Table 40–32 describes the syntax.

Table 40–32 ant Partition Composite Stop Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`partition`	The name of the partition.

Example 40–43 provides an example of stopping all composites in the partition named `myPartition`.

Example 40–43 Stopping All Composites in the Partition

```
ant -f ant-sca-mgmt.xml stopCompositesInPartition -Dhost=stabc10 -Dport=8001
-Dpartition=myPartition
```

40.7.5.2.23 Activating All Composites in the Partition Example 40–44 provides the syntax for activating all composites in the partition.

Example 40–44 Activating All Composites in the Partition

```
ant -f ant-sca-mgmt.xml activateCompositesInPartition -Dhost=host -Dport=port
-Duser=user -Dpartition=partition.name
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–33 describes the syntax.

Table 40–33 ant Partition Composite Activation Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`partition`	The name of the partition.

Example 40–45 provides an example of activating all composites in the partition named `myPartition`.

Example 40–45 Activating All Composites in the Partition

```
ant -f ant-sca-mgmt.xml activateCompositesInPartition -Dhost=stabc10 -Dport=8001
-Dpartition=myPartition
```

40.7.5.2.24 Retiring All Composites in the Partition Example 40–46 provides the syntax for retiring all composites in the partition.

Example 40–46 Retiring All Composites in the Partition

```
ant -f ant-sca-mgmt.xml retireCompositesInPartition -Dhost=host -Dport=port
 -Duser=user -Dpartition=partition.name
```

> **Note:** After specifying the user name, enter the password when prompted.

Table 40–34 describes the syntax.

Table 40–34 ant Partition Composite Retirement Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).
`port`	Port of the Oracle WebLogic Server (for example, `7001`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.
`partition`	The name of the partition.

Example 40–47 provides an example of retiring all composites in the partition named `myPartition`.

Example 40–47 Retiring All Composites in the Partition

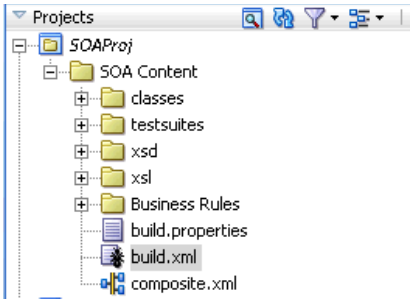
```
ant -f ant-sca-mgmt.xml retireCompositesInPartition -Dhost=stabc10 -Dport=8001
-Dpartition=myPartition
```

40.7.5.2.25 Upgrading a SOA Composite Application You can use `ant` to upgrade a SOA composite application from 10.1.3 to 11*g*. For information, see *Oracle Fusion Middleware Upgrade Guide for Oracle SOA Suite, WebCenter, and ADF*.

40.7.5.2.26 How to Manage SOA Composite Applications with ant Scripts The WebLogic Fusion Order Demo application provides an example of using `ant` scripts to compile, package, and deploy the application. You can create the initial `ant` build files by selecting **New** > **Ant** > **Buildfile from Project** from the **File** main menu.

Figure 40–26 shows the **build.properties** and **build.xml** files that display in the Application Navigator after creation.

Figure 40–26 ant Build Files



- **build.properties**

  A file that you edit to reflect your environment (for example, specifying Oracle home and Java home directories, setting server properties such as hostname and port number to use for deployment, specifying the application to deploy, and so on).

- **build.xml**

  Used by `ant` to compile, build, and deploy composite applications to the server specified in the **build.properties** file.

1. Modify the **build.properties** file to reflect your environment.
2. From the **Build** menu, select **Run Ant on** ***project_name***.

   This builds targets defined in the current project's build file.

40.7.6 Deploying SOA Composite Applications from Oracle Enterprise Manager Fusion Middleware Control

You can deploy SOA composite applications from Oracle Enterprise Manager Fusion Middleware Control. You must first create a deployable archive in Oracle JDeveloper or through the `ant` or WLST command line tools. The archive can consist of a single SOA composite application revision in a JAR file or multiple composite application revisions (known as a SOA bundle) in a ZIP file. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

40.7.7 Deploying SOA Composite Applications to a Cluster

You can deploy a SOA composite application into a clustered environment. For more information, see chapter "Configuring High Availability for Oracle Fusion Middleware SOA Suite" of the *Oracle Fusion Middleware High Availability Guide*.

40.8 Postdeployment Configuration

This section describes postdeployment configuration tasks.

40.8.1 Security

For information about securing SOA composite applications, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

40.8.2 Updating Connections

Ensure that any connections that you created to the application server or MDS repository are re-created to point to servers applicable to the next target environment. For more information, see Section 40.7.1.1.1, "Creating an Application Server Connection" and Section 40.7.3.2.1, "Creating a SOA-MDS Connection."

40.8.3 Updating Data Sources and Queues

Ensure that all JDBC data source, queue, and connection factory locations that you previously configured are applicable to the next target environment. For more information, see Section 40.5.1, "Creating Data Sources and Queues" and Section 40.5.2, "Creating Connection Factories and Connection Pooling."

40.8.4 Attaching Policies

You can attach policies to a deployed SOA composite application during runtime in Oracle Enterprise Manager Fusion Middleware Control. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

40.9 Testing and Troubleshooting

This section describes how to test and troubleshoot your SOA composite application.

40.9.1 Verifying Deployment

You can verify that you have successfully deployed your SOA composite application to the SOA Infrastructure. If successful, the deployed composite displays in the **Deployed Composites** tab of the SOA Infrastructure page of Oracle Enterprise Manager Fusion Middleware Control. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

40.9.2 Initiating an Instance of a Deployed Composite

You can initiate an instance of a deployed SOA composite application from the Test Instance page in Oracle Enterprise Manager Fusion Middleware Control. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

40.9.3 Automating the Testing of Deployed Composites

You can create, deploy, and run test cases that automate the testing of SOA composite applications. Test cases enable you to simulate the interaction between a SOA composite application and its web service partners before deployment in a production environment. You create test cases in Oracle JDeveloper and include them in a SOA composite application that is then deployed and administered from Oracle Enterprise Manager Fusion Middleware Control. You then run the test cases from Oracle Enterprise Manager Fusion Middleware Control.

For information about creating test cases, see Chapter 41, "Automating Testing of SOA Composite Applications."

For information about running test cases, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

40.9.4 Recompiling a Project After Receiving a Deployment Error

If you receive the error shown in Example 40–48 when deploying a SOA composite application from Oracle JDeveloper, recompile the project and redeploy the composite. This error is intermittent and should not occur again.

Example 40–48 Intermittent Deployment Error Message

```
Error deploying BPEL suitcase.
error while attempting to deploy the BPEL component file
"/scratch/aime1/work/mw9507/user_projects/domains/WLS_SOAWC/deployed-composites
/ManagementChainParticipantRuleComposite_rev1.0/sca_ManagementChainParticipantR
uleComposite_rev1.0/soa_59d10d76-08a5-41f0-ba89-32dcc2250002";
the exception reported is: java.lang.Exception: BPEL 1.1 compilation failed

This error contained an exception thrown by the underlying deployment module.
Verify the exception trace in the log (with logging level set to debug mode).

at
com.collaxa.cube.engine.deployment.DeploymentManager.deployComponent(Deployment
Manager.java:197)
at
com.collaxa.cube.ejb.impl.CubeServerManagerBean._deployOrLoadComponent(CubeServ
erManagerBean.java:820)
at
com.collaxa.cube.ejb.impl.CubeServerManagerBean.deployComponent(CubeServerManag
erBean.java:119)
```

40.9.5 Troubleshooting Common Deployment Errors

This section describes how to troubleshoot common deployment errors.

For information about general composite application troubleshooting issues, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

40.9.5.1 Common Oracle JDeveloper Deployment Issues

This section provides a list of common deployment issues to check.

- If you are deploying a single composite application, ensure that you are deploying from the **Project** menu. Right-click the project name in the Application Navigator, and select **Deploy** > ***SOA_profile_name***.

- If you are deploying multiple composite applications, ensure that you are deploying from the **Application** menu. (Right-click the application name in the Application Navigator, and select **Deploy** > ***SOA_bundle_profile_name***).
- Once you click Deploy and select the profile name, ensure that the Deployment Action page of the deployment wizard is displayed.
- Optionally enter a new revision ID (optional) and select the configuration plan (if any).
- If the composite application you are deploying is already located on the server with the same revision ID, then check the **Overwrite any existing composites with the same revision ID** checkbox in the Deploy Configuration page of the deployment wizard. Without selecting this option, deployment fails.
- If compilation fails, a compiler error occurred, and not a deployment error. You only see this error when you compile your project.
- If compiler messages are not obvious, check the compiler log. A link to this log file (`scac.log`) is displayed in the **Messages** tab. The message looks similar to that shown in Example 40–49.

Example 40–49 Compilation Log Message

```
Compilation of project 'FirstComposite.jpr' finished. Check '/scratch/myhome/
jdevWorkarea/mywork/Application11/FirstComposite/SCA-INF/classes/scac.log' for
details.
```

- After compilation is successful, an SAR/SOA bundle archive is built for the composite. For a SAR archive, the message shown in Example 40–50 is displayed in the **Deployment** tab.

Example 40–50 Archive Message

```
Wrote Archive Module to
/scratch/myhome/jdevWorkarea/mywork/Application11/FirstComposite/deploy/sca_
FirstComposite_rev1.0.jar
```

  For a SOA bundle archive, the message shown in Example 40–51 is displayed in the **Deployment** tab.

Example 40–51 Archive Message

```
Wrote Archive Module to
/scratch/myhome/jdevWorkarea/mywork/Application11/SecondComposite/deploy/sca_
SecondComposite_rev1.0.jar
Wrote Archive Module to
/scratch/myhome/jdevWorkarea/mywork/Application11/FirstComposite/deploy/sca_
FirstComposite_rev1.0.jar
Wrote Archive Module to
/scratch/myhome/jdevWorkarea/mywork/Application11/deploy/soabundle1.zip
```

- Ensure that all SAR file URLs look as follows

  `sca_`*`CompositeName`*`_rev`*`RevisionID`*`.jar`

  For example, `sca_FirstComposite_rev1.0.jar`.

- After this occurs, Oracle JDeveloper sends the archive binaries to the server. The following message is displayed in the **Deployment** tab. At this point, Oracle JDeveloper's deployment role ends and the server (SOA Infrastructure) takes control of deployment.

```
Deploying sca_FirstComposite_rev1.0.jar to myhost19:7001
```

- Upon successful deployment, you see the message shown in Example 40–52 in the **Deployment** tab.

Example 40–52 Successful Deployment Message

```
Received HTTP response from the server, response code=200 Successfully deployed
archive soa_bundle_name.zip to soa_server_name
```

- If deployment fails, the message shown in Example 40–53 is displayed in the **Deployment** tab with an error message (if any) from the server.

Example 40–53 Deployment Error Message

```
Error deploying the archive. Check server log for more details.
Connection refused.
Elapsed time for deployment: 8 seconds
```

- In most cases, the server provides some information about the error that occurred on the server. If you do not receive any error message from the server, then check `soa_server1-diagnostic.log` on the server to find additional information (where `soa_server1` is the name of the managed server). This file is located on the server in *domain_home*`/servers/soa_server1/logs`.

40.9.5.2 Common Configuration Plan Issues

This section provides a list of common configuration plan issues to check.

- If you selected a configuration plan to deploy, and it is not taking effect on the server, open the SAR file containing the configuration plan. You can find the file location from the **Deployment** tab in Oracle JDeveloper. Example 40–54 provides details.

Example 40–54 Archive Message

```
Wrote Archive Module to
/scratch/myhome/jdevWorkarea/mywork/Application11/FirstComposite/deploy/sca_
FirstComposite_rev1.0.jar
```

- Open the JAR file and ensure that it contains the `soaconfigplan.xml` file. This file is generated during deployment based on the configuration plan you selected.
- If this file is not present, try deploying the composite application again to ensure that you have correctly selected the configuration plan in the Deploy Configuration page of the deployment wizard.

40.9.5.3 Deploying to a Managed Oracle WebLogic Server

If you start a managed Oracle WebLogic Server without starting an Oracle WebLogic Administration Server (known as running in independence mode) and attempt to deploy a SOA composite application from Oracle JDeveloper, you receive the following error:

```
Deployment cannot continue! No SOA Configured target servers found
```

The Oracle WebLogic Administration Server must be running. Deployment uses the Oracle WebLogic Administration Server connection to identify the servers running Oracle SOA Suite. In addition, do not create an application server connection to a

managed Oracle WebLogic Server; only create connections to an Oracle WebLogic Administration Server.

You can also receive a similar error if the condition of the SOA-configured Oracle WebLogic Server is not healthy. This condition displays in the **Health** column of the Servers page of Oracle WebLogic Server Administration Console.

Note that you can use WLST to deploy SOA composite applications to a managed Oracle WebLogic Server without starting an Oracle WebLogic Administration Server. See Section 40.7.5.1, "How to Manage SOA Composite Applications with the WLST Utility" for details.

40.9.5.4 Deploying to a Two-Way, SSL-Enabled Oracle WebLogic Server

Deployment from Oracle JDeveloper to a two-way, SSL-enabled Oracle WebLogic Server is not supported.

40.9.5.5 Deploying with an Unreachable Proxy Server

You can receive an error similar to that shown in Figure 40–27 during SOA composite application deployment if you have a proxy server set in Oracle JDeveloper that is not reachable from your host.

Figure 40–27 Deployment Error Message

```
[09:56:25 AM] Sending internal deployment descriptor
[09:56:25 AM] Sending archive - sca_validationForCC_rev2.3.jar
[09:56:26 AM] Error sending deployment request to server soa_server1 [silverback:8001]
 java.net.UnknownHostException: emeacache.uk.oracle.com
[09:56:26 AM] ####  Deployment incomplete.  ####
[09:56:26 AM] Error deploying archive file:/C:/po/CreditCardValidation/validationForCC/
 (oracle.tip.tools.ide.fabric.deploy.common.SOARemoteDeployer)
```

A valid proxy setting is necessary for accessing a SOA Infrastructure (for example, `soa_server1`) outside the network. If the SOA Infrastructure is within the network, perform one of the following actions:

To change the proxy setting:

1. From the **Tools** menu, select **Preferences** > **Web Browser and Proxy**.
2. Perform one of the following tasks if the SOA server is within the network:
   a. Deselect **Use HTTP Proxy Server** if you can directly access the SOA Infrastructure without any proxy.
   b. In the **Exceptions** field, enter the hostname of the unreachable SOA server.

40.9.5.6 Releasing Locks to Resolve ADF Task Form EAR File Deployment Errors

If you deploy a SOA composite application JAR file and ADF task form EAR file, and the SOA JAR file is deployed successfully, but while deploying the EAR file, the following errors are displayed:

```
[wldeploy] weblogic.management.ManagementException: [Deployer:149163]The
domain edit lock is owned by another session in non-exclusive mode - this
deployment operation requires exclusive access to the edit lock and hence
cannot proceed. If you are using "Automatically Aquire Lock and Activate
Changes" in the console, then the lock will expire shortly so retry this
operation.
```

This means you must first release the lock from Oracle WebLogic Server Administration Console to successfully deploy the EAR file.

1. Log in to the Oracle WebLogic Server Administration Console.
2. Below the console banner at the top of the page, click **Preferences** > **User Preferences**.
3. Deselect **Automatically Acquire Lock and Activate Changes**.
4. Click **Save** and note that buttons such as **Lock and Edit** and **Release Configuration** are visible.

   Note the following description that is displayed in the Oracle WebLogic Server Administration Console:

   ```
   Automatically acquire the lock that enables configuration editing and
   automatically activate changes as the user modifies, adds and deletes items
    (for example, when the user clicks the 'Save' button). This feature is not
    available in production mode.
   ```

Note that this error can occur regardless of the deployment method you are using (for example, deploying through Oracle JDeveloper or through `ant` scripts).

40.9.5.7 Increasing Memory to Recover from Compilation Errors

If you receive out-of-memory errors during compilation of a SOA composite application, perform the following step to increase memory.

1. Under the `scac` element, increase the memory setting. For example:

   ```
   <jvmarg value="-Xmx512M"/>
   ```

41

Automating Testing of SOA Composite Applications

This chapter describes how to create, deploy, and run test cases that automate the testing of SOA composite applications. Test cases enable you to simulate the interaction between a SOA composite application and its web service partners before deployment in a production environment. This helps to ensure that a process interacts with web service partners as expected by the time it is ready for deployment to a production environment.

This chapter includes the following sections:

41.1 Introduction to the Composite Test Framework

Oracle SOA Suite provides an automated test suite framework for creating and running repeatable tests on a SOA composite application.

The test suite framework provides the following features:

- Simulates web service partner interactions
- Validates process actions with test data
- Creates reports of test results

41.1.1 Test Cases Overview

The test framework supports testing at the SOA composite application level. In this type of testing, wires, service binding components, service components (such as BPEL processes and Oracle Mediator service components), and reference binding components are tested.

For more information, see Section 41.3, "Creating Test Suites and Test Cases."

41.1.2 Test Suites Overview

Test suites consist of a logical collection of one or more test cases. Each test case contains a set of commands to perform as the test instance is executed. The execution

of a test suite is known as a test run. Each test corresponds to a single SOA composite application instance.

For more information, see the following:

- Section 41.3, "Creating Test Suites and Test Cases"
- Section 41.4, "Creating the Contents of Test Cases"

41.1.3 Emulations Overview

Emulations enable you to simulate the behavior of the following components with which your SOA composite application interacts during execution:

- Internal service components inside the composite
- Binding components outside the composite

Instead of invoking another service component or binding component, you can specify a response from the component or reference.

For more information, see the following:

- Section 41.2.2, "Emulations"
- Section 41.4, "Creating the Contents of Test Cases"

41.1.4 Assertions Overview

Assertions enable you to verify variable data or process flow. You can perform the following types of assertions:

- Entire XML document assertions:

  Compare the element values of an entire XML document to the expected element values. For example, compare the exact contents of an entire loan request XML document to another document. The `XMLTestCase` class in the `XMLUnit` package includes a collection of methods for performing assertions between XML files. For more information about these methods, visit the following URL:

  `http://xmlunit.sourceforge.net`

- Part section of message assertions:

  Compare the values of a part section of a message to the expected values. An example is a payload part of an entire XML document message.

- Nonleaf element assertions:

  Compare the values of an XML fragment to the expected values. An example is a loan application, which includes leaf elements SSN, email, customerName, and loanAmount.

- Leaf element assertions:

  Compare the value of a selected string or number element or a regular expression pattern to an expected value. An example is the SSN of a loan application.

For more information about asserts, see Section 41.2.3, "Assertions."

41.2 Introduction to the Components of a Test Suite

This section describes and provides examples of the test components that comprise a test case. Methods for creating and importing these tests into your process are described in subsequent sections of this chapter.

41.2.1 Process Initiation

You first define the operation of your process in a binding component service such as a SOAP web service. Example 41–1 defines the operation of `initiate` to initiate the `TestFwk` SOA composite application. The initiation payload is also defined in this section:

Example 41–1 Process Initiation

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [6/13/07 10:50 AM]. -->
<compositeTest compositeDN="TestFwk"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <content>
          <loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
            <SSN>111222333</SSN>
            <email>joe.smith@oracle.com</email>
            <customerName>Joe Smith</customerName>
            <loanAmount>20000</loanAmount>
            <carModel>Camry</carModel>
            <carYear>2007</carYear>
            <creditRating>800</creditRating>
          </loanApplication>
        </content>
      </part>
    </message>
  </initiate>
</compositeTest>
```

41.2.2 Emulations

You create emulations to simulate the message data that your SOA composite application receives from web service partners.

In the test code in Example 41–2, the loan request is initiated with an error. A fault message is received in return from a web service partner:

Example 41–2 Emulations

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/3/07 3:29 PM]. -->
<compositeTest compositeDN="CompositeTest"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
```

```
  </initiate>
  <wireActions wireSource="LoanBroker/CreditRatingService" operation="process">
    <emulate duration="PT0S">
      <fault faultName="ser:NegativeCredit" xmlns:ser="http://services.otn.com">
        <message>
          <part partName="payload">
            <filePath>creditRatingFault.xml</filePath>
          </part>
        </message>
      </fault>
    </emulate>
  </wireActions>
</compositeTest>
```

Two message files, `loanApplication.xml` and `creditRatingFault.xml`, are invoked in this emulation. If the loan application request in `loanApplication.xml` contains a social security number beginning with `0`, the `creditRatingFault.xml` file returns the fault message shown in Example 41–3:

Example 41–3 Fault Message

```
<error xmlns="http://services.otn.com">
  Invalid SSN, SSN cannot start with digit '0'.
</error>
```

For more information, see Section 41.4, "Creating the Contents of Test Cases."

41.2.3 Assertions

You create assertions to validate an entire XML document, a part section of a message, a nonleaf element, or a leaf element at a point during SOA composite application execution. Example 41–4 instructs Oracle SOA Suite to ensure that the content of the `customername` variable matches the content specified.

Example 41–4 Assertions

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [6/13/07 10:51 AM]. -->
<compositeTest compositeDN="TestFwk"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <content>
          <loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
            <SSN>111222333</SSN>
            <email>joe.smith@oracle.com</email>
            <customerName>Joe Smith</customerName>
            <loanAmount>20000</loanAmount>
            <carModel>Camry</carModel>
            <carYear>2007</carYear>
            <creditRating>800</creditRating>
          </loanApplication>
        </content>
      </part>
    </message>
  </initiate>
  <wireActions wireSource="client" operation="initiate">
    <assert comparisonMethod="string">
```

```
      <expected>
        <location key="input" partName="payload"
 xpath="/s1:loanApplication/s1:customerName"
 xmlns:s1="http://www.autoloan.com/ns/autoloan"/>
        <simple>Joe Smith</simple>
      </expected>
    </assert>
  </wireActions>
</compositeTest>
```

For more information, see Section 41.4, "Creating the Contents of Test Cases."

41.2.4 Message Files

Message instance files provide a method for simulating the message data received back from web service partners. You can manually enter the received message data into this XML file or load a file through the test mode of the SOA Composite Editor. For example, the following message file simulates a credit rating result of `900` returned from a partner:

```
<rating xmlns="http://services.otn.com">900</rating>
```

For more information about loading message files into test mode, see Section 41.4, "Creating the Contents of Test Cases."

41.3 Creating Test Suites and Test Cases

This section describes how to create test suites and their test cases for a SOA composite application. The test cases consist of sets of commands to perform as the test instance is executed.

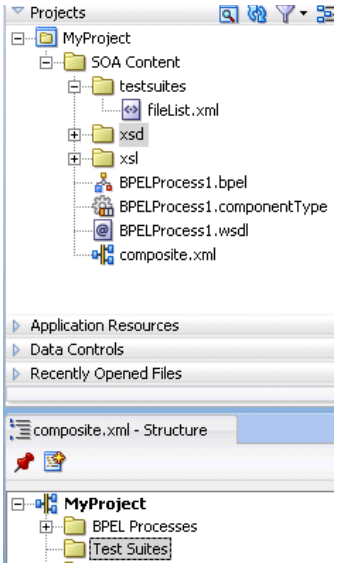
Note: Do *not* enter a multibyte character string as a test suite name or test case name. Doing so causes an error to occur when the test is executed from Oracle Enterprise Manager Fusion Middleware Control.

41.3.1 How to Create Test Suites and Test Cases

To create test suites and test cases:

1. Open the SOA Composite Editor.
2. Open the SOA composite application in which to create a test suite.
3. Go to the Application Navigator or Structure window. If the Structure window shown in Figure 41–1 does not appear, select **Structure** from the **View** main menu.

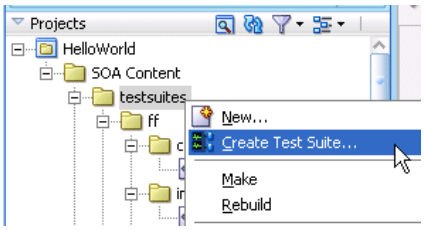
Figure 41–1 Structure Window



4. Create a test suite in either of two ways:

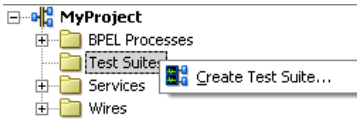
   a. In the Application Navigator, right-click **testsuites** and select **Create Test Suite**. Figure 41–2 provides details.

Figure 41–2 Create Test Suite Selection



   b. In the Structure window, right-click **Test Suites** and select **Create Test Suite**. Figure 41–3 provides details.

Figure 41–3 Create Test Suite Selection



5. Enter a test suite name (for example, `OrderBookingMainTestsuite` of the Fusion Order Demo).

6. Click **OK**.

   The Create Composite Test dialog appears.

7. Enter a test name (for this example, `NoErrorSanityTest` of the Fusion Order Demo is entered) and an optional description. This description displays in the **Description** column of the Test Cases page of the **Unit Tests** tab in Oracle Enterprise Manager Fusion Middleware Control.

8. Click **OK**.

This action creates a test named **NoErrorSanityTest.xml** in the Application Navigator, along with the following subfolders:

- **componenttests**

  This folder is not used in 11*g* Release 1.

- **includes**

  This folder is not used in 11*g* Release 1.

- **messages**

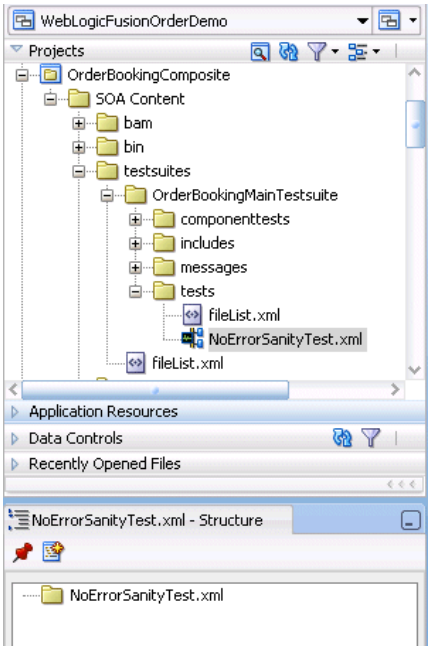
  Contains message test files that you load into this directory through the test mode user interface.

- **tests**

  Contains **NoErrorSanityTest.xml**.

A **NoErrorSanityTest.xml** folder also displays in the Structure window. Figure 41–4 provides details. This indicates that you are in the test mode of the SOA Composite Editor. You can create test initiations, assertions, and emulations in test mode. No other modifications, such as editing the property dialogs of service components or dropping service components into the editor, can be performed in test mode.

Figure 41–4 NoErrorSanityTest.xml Folder



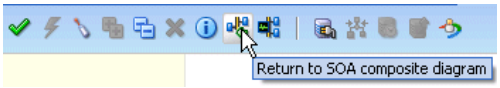
The following operating system test suite directory is also created:

`C:\JDeveloper\mywork\`*`application_name`*`\`*`project_name`*`\testsuites\OrderBookingMainTestsuite`

The following subdirectories for adding test files are created beneath `OrderBookingMainTestsuite`: `componenttests`, `includes`, `messages`, and `tests`.

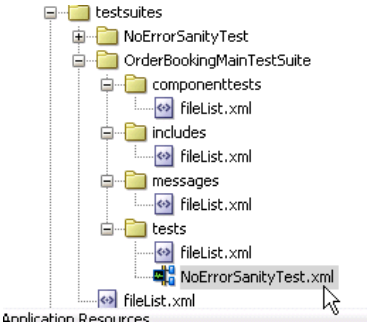
9. If you want to exit test mode and return to design mode in the SOA Composite Editor, click the last icon below **NoErrorSanityTest.xml** above the designer. Figure 41–5 provides details.

Figure 41–5 Test Mode Exit



10. Save your changes when prompted.
11. Under the **testsuites** folder in the Application Navigator, double-click **NoErrorSanityTest.xml** to return to test mode. Figure 41–6 provides details.

Figure 41–6 Test Mode Access



Notes:

- Do not edit the **filelist.xml** files that display under the subfolders of the **testsuites** folder. These files are automatically created during design time, and are used during runtime to calculate the number of test cases.
- You cannot create test suites within other test suites. However, you can organize a test suite into subdirectories.

The Fusion Order Demo provides examples of using test suites. For more information about the Fusion Order Demo, see Chapter 3, "Introduction to the SOA Sample Application."

41.4 Creating the Contents of Test Cases

Test cases consist of process initiations, emulations, and assertions. You add these actions to test cases in the test mode of the SOA Composite Editor. You create process initiations to initiate client inbound messages into your SOA composite application. You create emulations to simulate input or output message data, fault data, callback data, or all of these types that your SOA composite application receives from web service partners. You create assertions to validate entire XML documents, part sections of messages, nonleaf elements, and leaf elements as a process is executed.

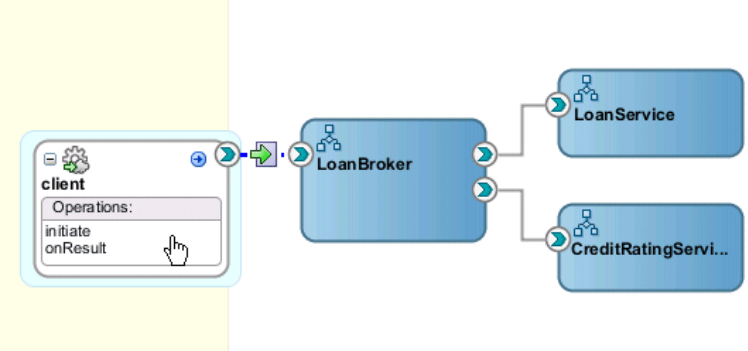
41.4.1 How to Initiate Inbound Messages

To initiate inbound messages:

You must first initiate the sending of inbound client messages to the SOA composite application.

1. Go to the SOA Composite application in test mode.
2. Double-click the service binding component shown in Figure 41–7 (for this example, named **initiate**).

Figure 41–7 Binding Component Service Access



The Edit Initiate dialog appears.

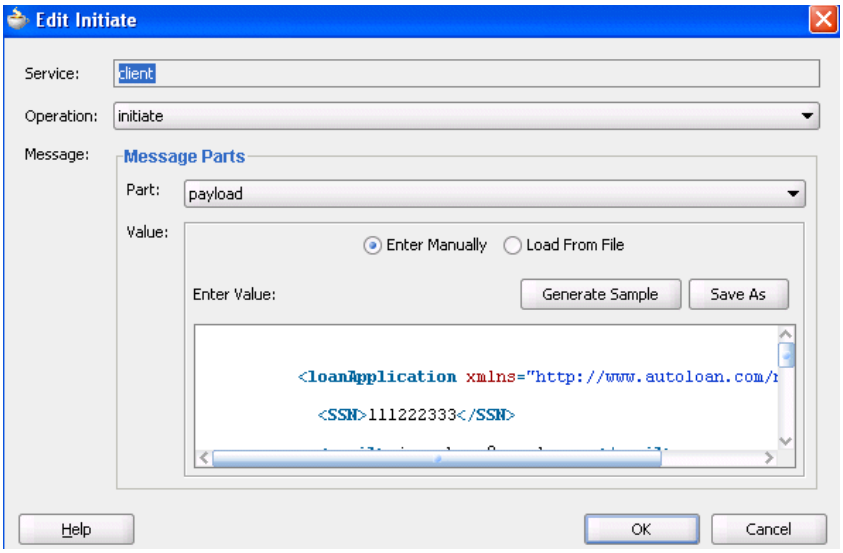
3. Enter the details shown in Table 41–1:

Table 41–1 Edit Initiate Dialog Fields and Values

Field	Value
Service	Displays the name of the binding component service (**client**).
Operation	Displays the operation type of the service binding component (**initiate**).
Part	Select the type of inbound message to send (for example, **payload**).
Value	Create a simulated message to send from a client:
▪ **Enter Manually**	Click to manually enter message data in the **Enter Value** field. A **Generate Sample** button enables you to automatically generate a sample file for testing. Click **Save As** to save the sample file.
▪ **Load From File**	Click the **Browse** icon to load message data from a file. The file is added to the **messages** folder in the Application Navigator.

Figure 41–8 shows this dialog:

Figure 41–8 Edit Initiate Dialog



Example 41–5 shows an inbound process initiation message from a client:

Example 41–5 Inbound Process Initiation Message

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/12/07 8:36 AM]. -->
<compositeTest compositeDN="CompositeTest"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about/>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
. . .
. . .
```

The `loanApplication.xml` referenced in the process initiation file contains a loan application payload. Example 41–6 provides details.

Example 41–6 Loan Application Payload

```
<loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
  <SSN>111222333</SSN>
  <email>joe.smith@oracle.com</email>
  <customerName>Joe Smith</customerName>
  <loanAmount>20000</loanAmount>
  <carModel>Camry</carModel>
  <carYear>2007</carYear>
  <creditRating>800</creditRating>
</loanApplication>
```

4. Click **OK**.

41.4.2 How to Emulate Outbound Messages

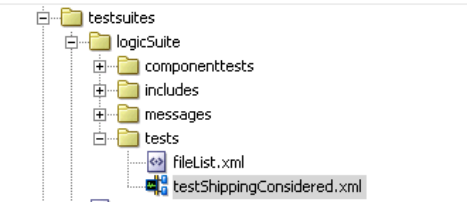
To emulate outbound messages:

> **Note:** The creation of multiple emulations in an instance in a test case is supported only if one emulation is for an output message and the other is for a callback message.

You can simulate a message returned from a synchronous web service partner.

1. Go to the SOA composite application in test mode.
2. Beneath the **testsuites** folder in the Application Navigator, double-click a test case. Figure 41–9 provides details.

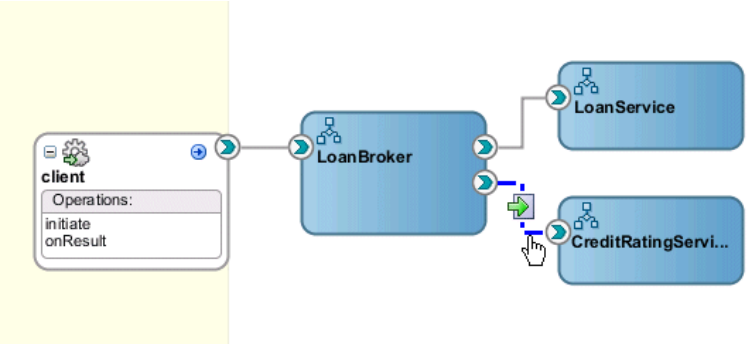
Figure 41–9 Test Case Access



The SOA composite application in the SOA Composite Editor is refreshed to display in test mode. This mode enables you to define test information.

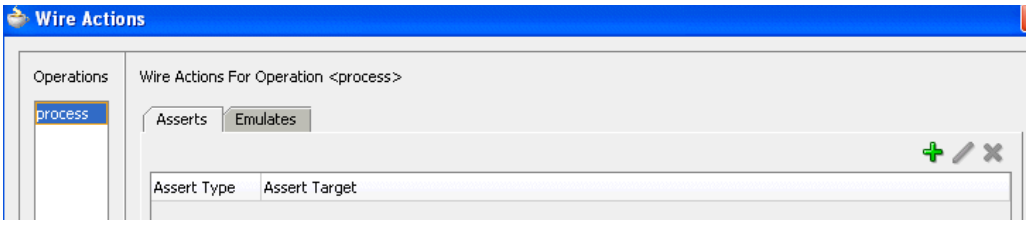
3. Double-click the wire of the SOA composite application area to test. For the example shown in Figure 41–10, the wire between the **LoanBroker** process and the synchronous **CreditRating** web service is selected.

Figure 41–10 Wire Access



This displays the Wire Actions dialog shown in Figure 41–11, from which you can design emulations and assertions for the selected part of the SOA composite application.

Figure 41–11 Wire Actions Dialog



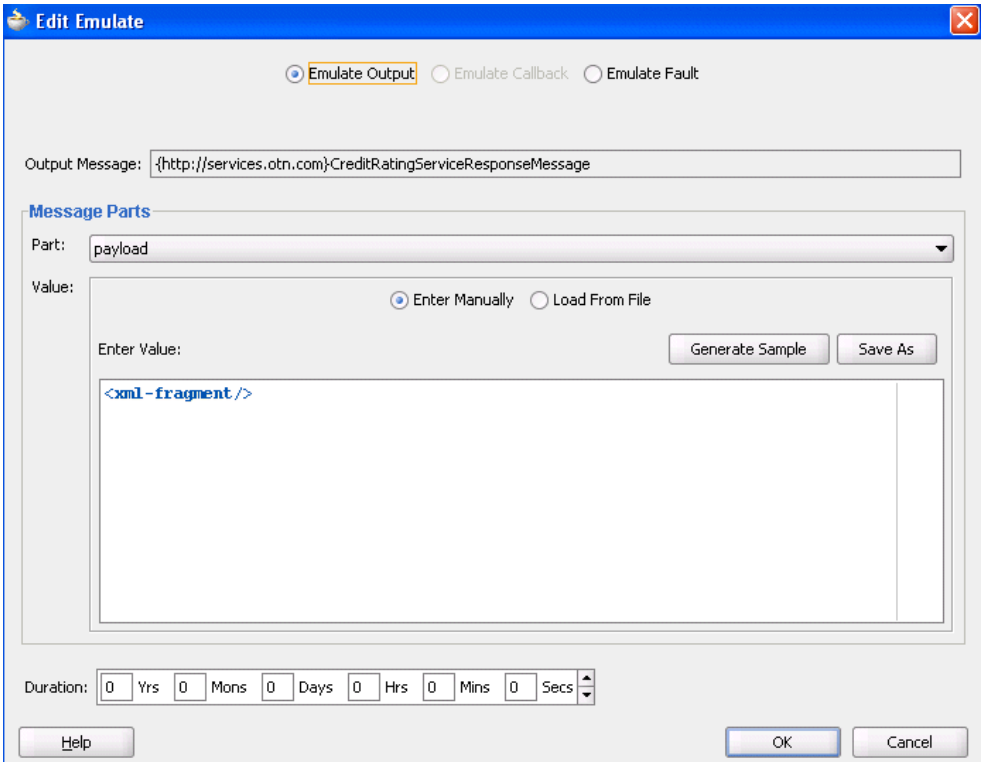
4. Click the **Emulates** tab.
5. Click the **Add** icon.
6. Click **Emulate Output**.
7. Enter the details described in Table 41–2:

Table 41–2 Emulate Output Message Dialog Fields and Values

Field	Value
Part	Select the message part containing the output (for example, **payload**).
Value	Create a simulated output message to return from a web service partner:
▪ **Enter Manually**	Click to manually enter message data in the **Enter Value** field. A **Generate Sample** button enables you to automatically generate a sample file for testing. Click **Save As** to save the sample file.
▪ **Load From File**	Click the **Browse** icon to load message data from a file. The file is added to the **messages** folder in the Application Navigator.
Duration	Enter the maximum amount of time to wait for the message to be delivered from the web service partner.

Figure 41–12 shows this dialog:

Figure 41–12 Emulate Dialog with Emulate Output Selected



Example 41–7 shows a simulated output message from a synchronous web service partner that you enter manually or load from a file:

Example 41–7 Simulated Output Message Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/3/07 3:26 PM]. -->
<compositeTest compositeDN="CompositeTest"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
  <wireActions wireSource="LoanBroker/CreditRatingService" operation="process">
    <emulate duration="PT0S">
      <message>
        <part partName="payload">
          <filePath>creditRatingResult.xml</filePath>
        </part>
      </message>
    </emulate>
  </wireActions>
</compositeTest>
```

The `creditRatingResult.xml` message file referenced in the output message provides details about the credit rating result.

```
<rating xmlns="http://services.otn.com">900</rating>
```

8. Click **OK**.

41.4.3 How to Emulate Callback Messages

To emulate callback messages:

> **Note:** The creation of multiple emulations in an instance in a test case is supported only if one emulation is for an output message and the other is for a callback message.

You can simulate a callback message returned from an asynchronous web service partner.

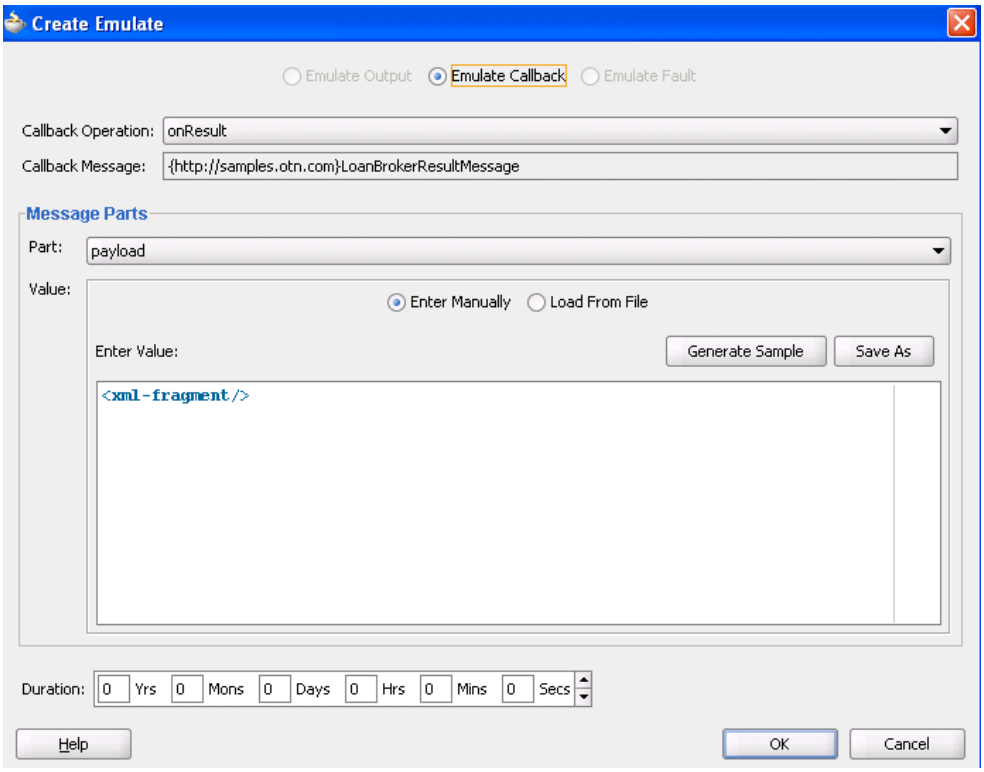
1. Access the Wire Actions dialog by following Step 1 through Step 3 of Section 41.4.2, "How to Emulate Outbound Messages."
2. Click the **Emulates** tab.
3. Click the **Add** icon.
4. Click **Emulate Callback**. This field is only enabled for asynchronous processes.
5. Enter the details described in Table 41–3:

Table 41–3 Emulate Callback Message Fields

Field	Value
Callback Operation	Select the callback operation (for example, **onResult**).
Callback Message	Displays the callback message name of the asynchronous process.
Part	Select the message part containing the callback (for example, **payload**).
Value	Create a simulated callback message to return from an asynchronous web service partner:
▪ **Enter Manually**	Click to manually enter message data in the **Enter Value** field. A **Generate Sample** button enables you to automatically generate a sample file for testing. Click **Save As** to save the sample file.
▪ **Load From File**	Click the **Browse** icon to load message data from a file. The file is added to the **messages** folder in the Application Navigator.
Duration	Enter the maximum amount of time to wait for the callback message to be delivered from the web service partner.

Figure 41–13 shows this dialog:

Figure 41–13 Emulate Dialog with Emulate Callback Selected



Example 41–8 shows a simulated callback message from a web service partner. You enter this message manually or load it from a file:

Example 41–8 Simulated Callback Message Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/3/07 3:27 PM]. -->
<compositeTest compositeDN="CompositeTest"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
  <wireActions wireSource="LoanBroker/LoanService" operation="initiate">
    <emulate callbackOperation="onResult" duration="PT0S">
      <message>
        <part partName="payload">
          <filePath>loanOffer.xml</filePath>
        </part>
      </message>
    </emulate>
  </wireActions>
</compositeTest>
```

The `loanOffer.xml` message file referenced in the callback message provides details about the credit rating approval. Example 41–9 provides details.

Example 41–9 Credit Rating Approval Details

```
<loanOffer xmlns="http://www.autoloan.com/ns/autoloan">
  <providerName>Bank Of America</providerName>
  <selected>false</selected>
  <approved>true</approved>
  <APR>1.9</APR>
</loanOffer>
```

6. Click **OK**.

41.4.4 How to Emulate Fault Messages

To emulate fault messages:

You can simulate a fault message returned from a web service partner. This simulation enables you to test fault handling capabilities in your process.

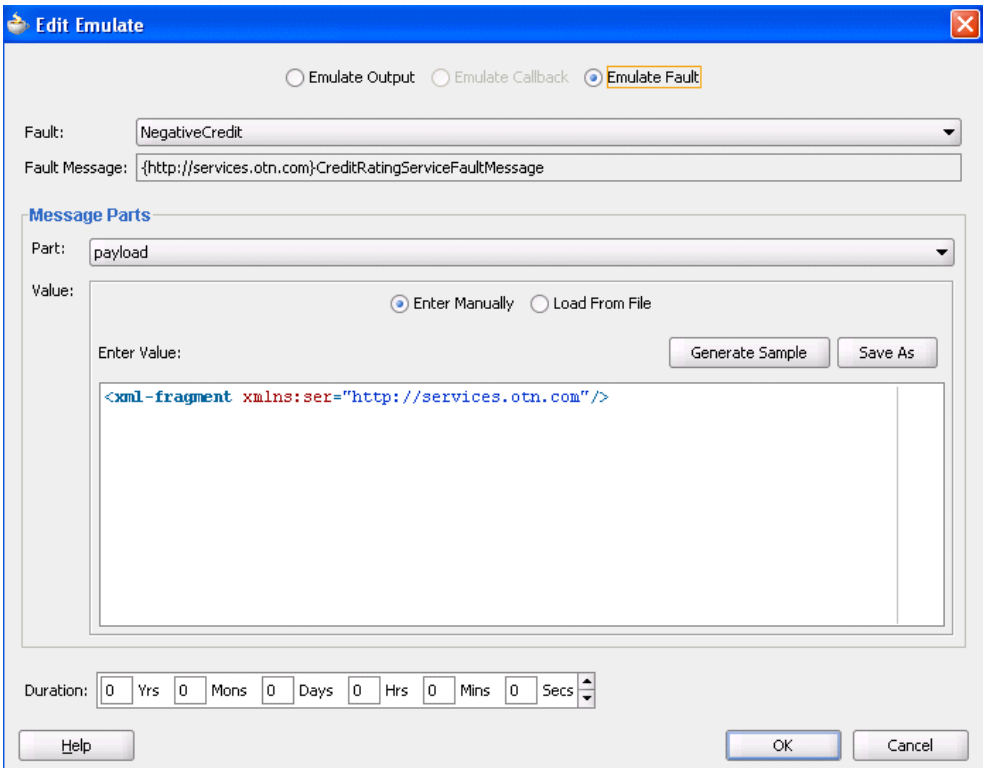
1. Access the Wire Actions dialog by following Step 1 through Step 3 of Section 41.4.2, "How to Emulate Outbound Messages."
2. Click the **Emulates** tab.
3. Click the **Add** icon.
4. Click **Emulate Fault**.
5. Enter the details described in Table 41–4:

Table 41–4 Emulate Fault Message Fields

Field	Value
Fault	Select the fault type to return from a partner (for example, **NegativeCredit**).
Fault Message	Displays the message name.
Part	Select the message part containing the fault (for example, **payload**).
Value	Create a simulated fault message to return from a web service partner:
▪ **Enter Manually**	Click to manually enter message data in the **Enter Value** field. A **Generate Sample** button enables you to automatically generate a sample file for testing. Click **Save As** to save the sample file.
▪ **Load From File**	Click the **Browse** icon to load message data from a file. The file is added to the **messages** folder in the Application Navigator.
Duration	Enter the maximum amount of time to wait for the fault message to be delivered from the web service partner.

Figure 41–14 shows this dialog:

Figure 41–14 Emulate Dialog with Emulate Fault Selected



An example of a simulated fault message from a web service partner that you enter manually or load from a file is shown in Section 41.2.2, "Emulations."

6. Click **OK**.

41.4.5 How to Create Assertions

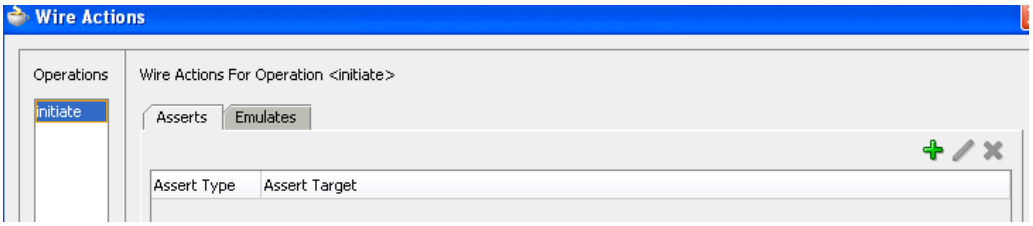
To create assertions:

You perform assertions to verify variable data or process flow. Assertions enable you to validate test data in an entire XML document, a part section of a message, a nonleaf element, or a leaf element as a process is executed. This is done by extracting a value and comparing it to an expected value.

1. Access the Wire Actions dialog by following Step 1 through Step 3 of Section 41.4.2, "How to Emulate Outbound Messages."
2. Click the **Asserts** tab.

   Figure 41–15 shows this dialog:

Figure 41–15 Wire Actions Dialog with Asserts Tab Selected



3. Click the **Add** icon.

   The Create Assert dialog appears.

4. Select the type of assertion to perform at the top of the dialog, as shown in Table 41–5. If the operation supports only input messages, the **Assert Input** button is enabled. If the operation supports both input and output messages, the **Assert Input** and **Assert Output** buttons are both enabled.

Table 41–5 Assertion Types

Type	Description
Assert Input	Select to create an assertion in the inbound direction.
Assert Output	Select to create an assertion in the outbound direction.
Assert Callback	Select to create an assertion on a callback.
Assert Fault	Select to assert a fault into the application flow.

5. See the section shown in Table 41–6 based on the type of assertion you want to perform.

Table 41–6 Assertion Types

For an Assertion on...	See...
▪ A part section of a document ▪ A nonleaf element ▪ An entire XML document	Section 41.4.5.1, "Creating Assertions on a Part Section, Nonleaf Element, or Entire XML Document"
A leaf element	Section 41.4.5.2, "Creating Assertions on a Leaf Element"

41.4.5.1 Creating Assertions on a Part Section, Nonleaf Element, or Entire XML Document

To create assertions on a part section, nonleaf element, or entire XML document:

This test compares the values to the expected values.

> **Note:** If the message contains multiple parts (for example, **payload1**, **payload2**, and **payload3**), you must create separate assertions for each part.

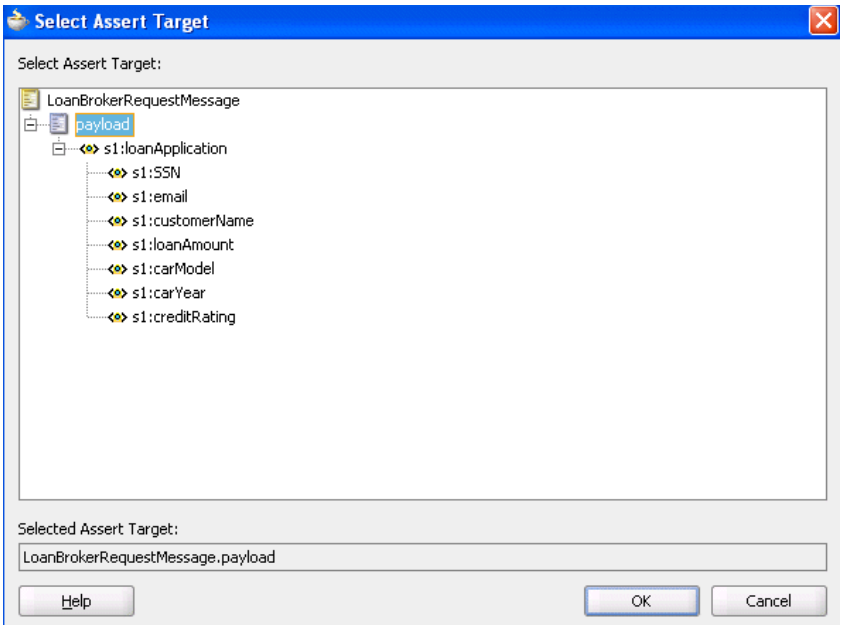
1. Click **Browse** to select the target part section, nonleaf element, or entire XML document to assert.

   The Select Assert Target dialog appears.

2. Select a value, and click **OK**. For example, select a variable such as **payload** to perform a part section assertion.

   Figure 41–16 shows this dialog. While this example shows how to perform a part section assertion, selecting **LoanBrokerRequestMessage** is an example of an entire XML document assertion and selecting **loanApplication** is an example of a nonleaf assertion.

Figure 41–16 Select a Part Section of a Message



The Create Assert dialog refreshes based on your selection of a variable.

3. Enter details in the remaining fields, as shown in Table 41–7:

Table 41–7 Create Assert Dialog Fields and Values

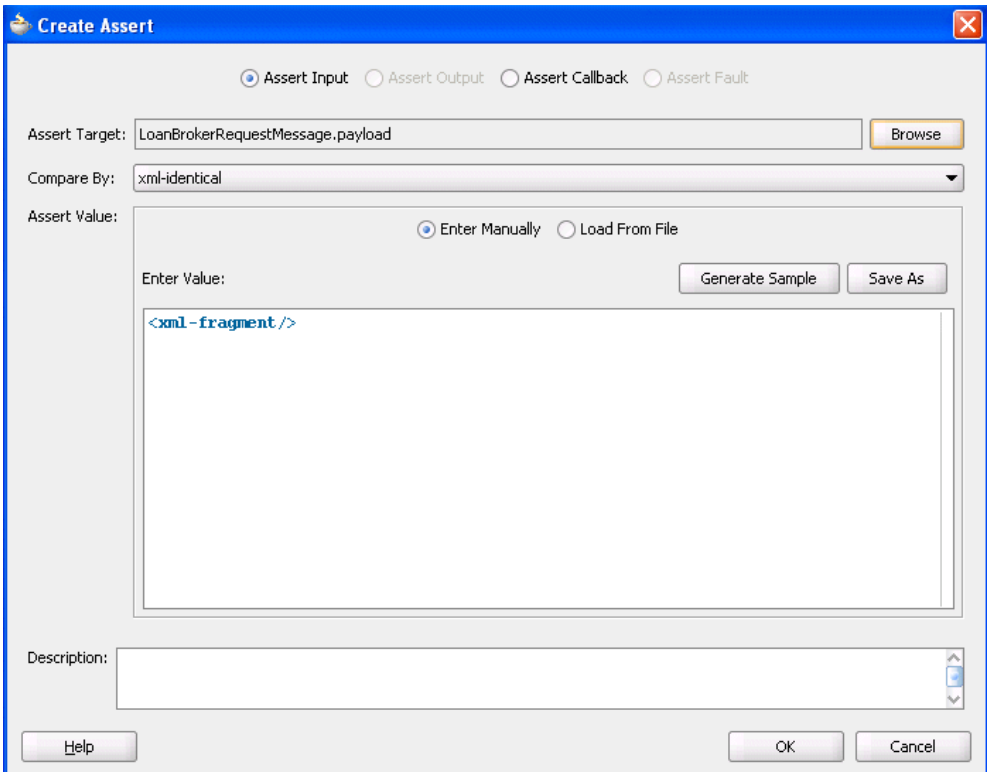
Field	Value
Fault	Select the type of fault to assert (for example, **NegativeCredit**). This field only displays if you select **Assert Fault** in Step 4.of Section 41.4.5, "How to Create Assertions."
Assert Target	Displays the assert target you selected in Step 2.
Compare By	Specify the strictness of the comparison. ▪ **xml-identical**: Used when the comparison between the elements and attributes of the XML documents must be exact. If there is any difference between the two XML documents, the comparison fails. For example, the comparison fails if one document uses an element name of `purchaseOrder`, while the other uses an element name of `invoice`. The comparison also fails if the child attributes of two elements are the same, but the attributes are ordered differently in each element. ▪ **xml-similar**: Used when the comparison must be similar in content, but does not need to exactly match. For example, the comparison succeeds if both use the same namespace URI, but have different namespace prefixes. The comparison also succeeds if both contain the same element with the same child attributes, but the attributes are ordered differently in each element. In both of these examples, the differences are considered recoverable, and therefore similar. For more information about comparing the contents of XML files, see the XMLUnit web site: `http://xmlunit.sourceforge.net/userguide/html/ar01s03.html#The%20Difference%20Engine`

Table 41–7 (Cont.) Create Assert Dialog Fields and Values

Field	Value
Part	Select the message part containing the XML document (for example, **payload**).
Value	Create an XML document whose content is compared to the assert target content:
▪ **Enter Manually**	Click to manually enter message data in the **Enter Value** field. A **Generate Sample** button enables you to automatically generate a sample file for testing. Click **Save As** to save the sample file.
▪ **Load From File**	Click the **Browse** icon to load message data from a file. The file is added to the **messages** folder in the Application Navigator.
Description	Enter an optional description.

Figure 41–17 shows this dialog with **Assert Input** selected:

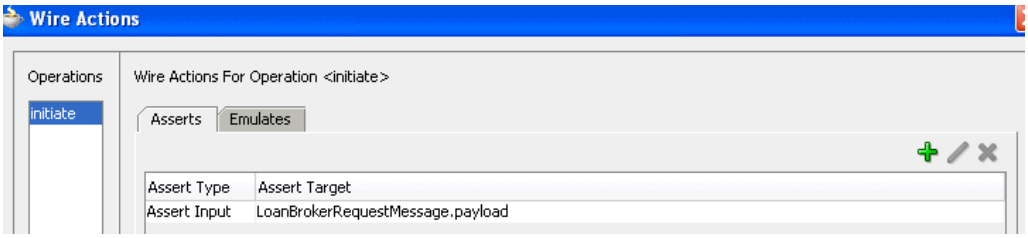
Figure 41–17 Create Assert Dialog with Assert Input Selected



4. Click **OK**.

   The Wire Actions dialog shown in Figure 41–18 displays your selection.

Figure 41–18 Wire Actions Dialog with Asserts Tab Selected



5. Click **OK**.

41.4.5.2 Creating Assertions on a Leaf Element

To create assertions on a leaf element:

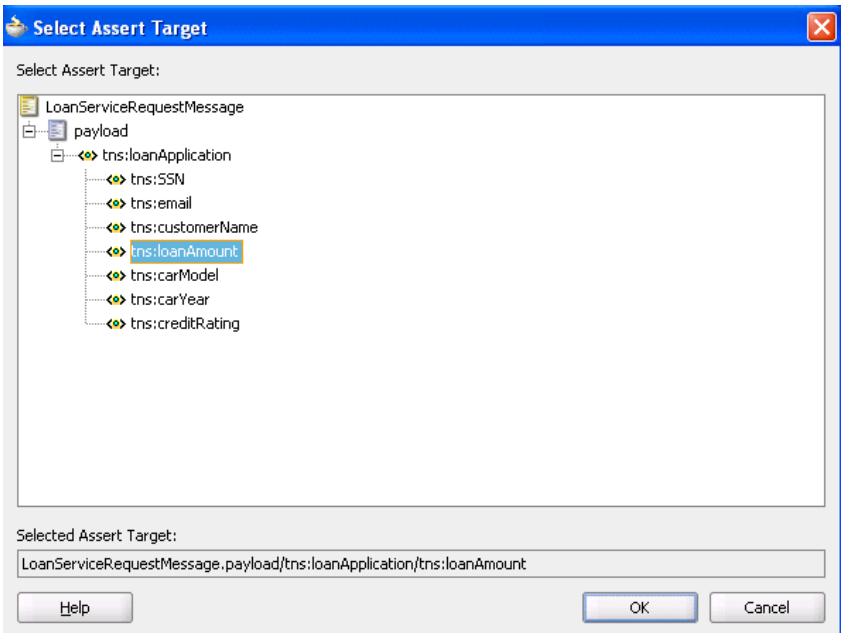
This test compares the value to an expected value.

1. Click **Browse** to select the leaf element to assert.

   The Select Assert Target dialog appears.

2. Select a leaf element, and click **OK**. For example, select **loanAmount** to perform an assertion. Figure 41–19 provides details.

Figure 41–19 Selection of a Leaf Element



   The Create Assert dialog refreshes based on your selection of an entire XML document.

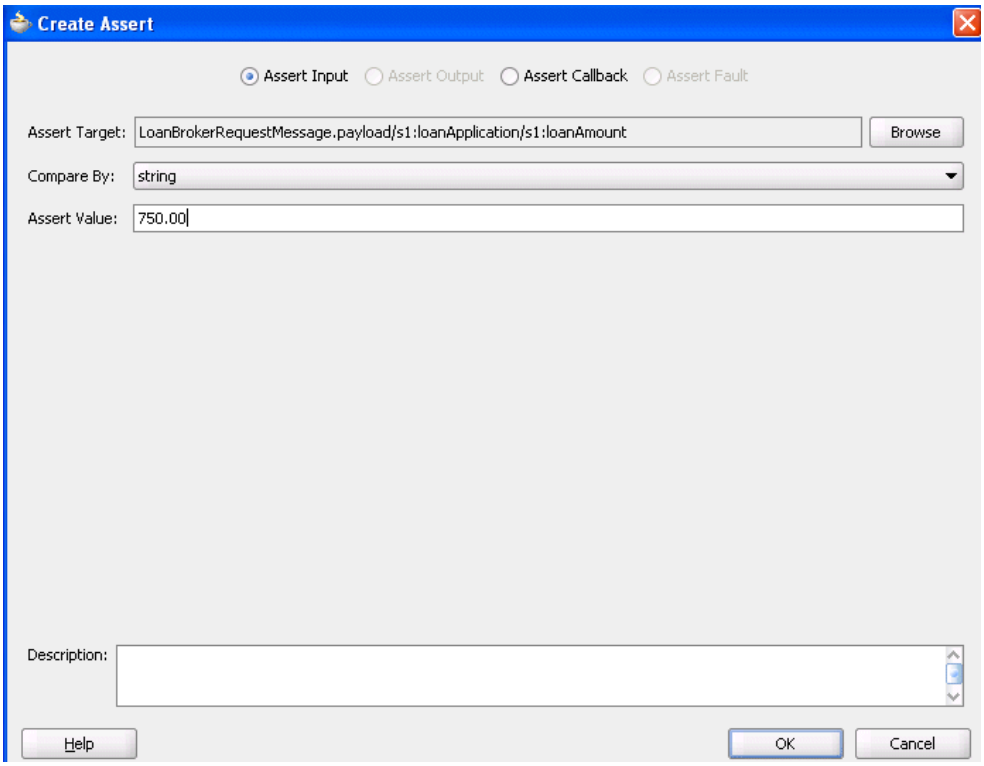
3. Enter details in the remaining fields, as shown in Table 41–8:

Table 41–8 Create Assert Dialog Fields and Values

Field	Value
Fault	Select the type of fault to assert (for example, **NegativeCredit**). This field only displays if you select **Assert Fault** in Step 4 of Section 41.4.5, "How to Create Assertions."
Callback Operation	Select the type of callback to assert (for example, **onResult**). This field only displays if you select **Assert Callback** in Step 4 of Section 41.4.5, "How to Create Assertions."
Assert Target	Displays the variable assert target you selected in Step 2.
Compare By	Select the type of comparison: ▪ **string**: Compares string values ▪ **number**: Compares numeric values ▪ **pattern-match**: Compares a regular expression pattern (for example, `[0-9]*`). Java Development Kit (JDK) regular expression (regexp) constructs are supported. For example, entering a pattern of `ab[0-9]*cd` means that a value of `ab123cd` or `ab456cd` is correct. An asterisk (*) indicates any number of occurrences.
Assert Value	Enter the value you are expecting. This value is compared to the value for the assert target.
Description	Enter an optional description.

Figure 41–20 shows this dialog with **Assert Input** selected:

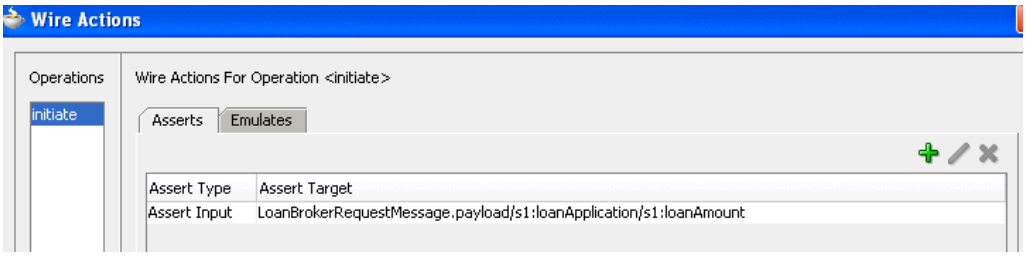
Figure 41–20 Create Assert Dialog



4. Click **OK**.

The Wire Actions dialog shown in Figure 41–21 displays your selection.

Figure 41–21 Wire Actions Dialog with Asserts Tab Selected



41.4.6 What You May Need to Know About Assertions

When a test is executed, and the response type returned is different from the type expected, the assertion is skipped. For example, you are expecting a fault (`RemoteFault`) to be returned for a specific message, but a response (`BpelResponseMessage`) is instead returned.

As a best practice, always assert and emulate the expected behavior.

41.5 Deploying and Running a Test Suite

After creating a test suite of test cases, you deploy the suite as part of a SOA composite application. You then run the test suites from Oracle Enterprise Manager Fusion Middleware Control.

- For information about deploying a SOA composite application from Oracle JDeveloper., see Section 40.7.1.1, "How to Deploy a Single SOA Composite."
- For information about deploying a SOA composite application and running a test suite from Oracle Enterprise Manager Fusion Middleware Control, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.
- For information about using the `sca_test` WLST command to execute a test suite, see Section "sca_test" of *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.
- For information about using the `ant-sca-test.xml` ant script to execute a test suite, see Section 40.7.5.2.1, "Testing a SOA Composite Application."

Part IX

Advanced Topics

This part describes advanced topics.

This part contains the following chapters:

42

Managing Large Documents and Large Numbers of Instances

This chapter describes the best practices for managing large documents and metadata and for managing environments with large numbers of instances in Oracle SOA Suite.

This chapter includes the following sections:

- Section 42.1, "Best Practices for Handling Large Documents"
- Section 42.2, "Best Practices for Handling Large Metadata"
- Section 42.3, "Best Practices for Handling Large Numbers of Instances"

For more information about Oracle SOA Suite tuning and performance, see *Oracle Fusion Middleware Performance and Tuning Guide*.

42.1 Best Practices for Handling Large Documents

This section describes the following scenarios for handling large documents and the best practice approach for each scenario. Oracle recommends that you follow these best practices before developing and executing large payloads.

42.1.1 Use Cases for Handling Large Documents

This section describes use cases for handling large documents.

42.1.1.1 Passing Binary Objects as Base64-Encoded Text in XML Payloads

This section describes use cases for passing binary objects as Base64-encoded text in the XML payload.

42.1.1.1.1 SOAP Inline In this use case, the binary attachments (for example, an image) are Base64-encoded as text and then passed inline in the XML document. Table 42–1 provides details.

Table 42–1 Capabilities

Capability	Description
Security	Supported.
Filter/Transformation/Assign	Use of transformations may lead to slower performance, out-of-memory errors, or both.
Fanout	Supported.
Binding	WS binding sends it as a document object model (DOM).

Table 42–1 (Cont.) Capabilities

Capability	Description
Oracle BPEL Process Manager/Oracle Mediator	Can be decoded in a BPEL process using Java `exec`.

42.1.1.1.2 SOAP MTOM In this use case, the binary attachments (for example, an image) are Base64-encoded as text and then passed as a Message Transmission Optimization Mechanism (MTOM) document. Table 42–2 provides details.

Table 42–2 Capabilities

Capability	Description
Security	Supported.
Filter/Transformation/Assign	Assign activities are supported.
Fanout	Supported.
Binding	WS binding materializes the attachment sent as MTOM and puts it inside in Base64-encoded format (streaming is not supported). Outbound MTOM is not supported.
Oracle BPEL Process Manager/Oracle Mediator	No additional work is required.

42.1.1.1.3 Opaque Passed by File/FTP Adapters In this use case, the binary attachments (for example, an image) are Base64-encoded as text and then passed inline in the XML document. Table 42–3 provides details.

Table 42–3 Capabilities

Capability	Description
Security	Not supported.
Filter/Transformation/Assign	Pass through.
Fanout	Supported.
Binding	Adapter encodes it to Base64 format.
Oracle BPEL Process Manager/Oracle Mediator	Supported. Opaque content cannot be manipulated in an assign or a transformation activity.

42.1.1.1.4 Opaque Passed by Oracle B2B In this use case, the binary attachments (for example, an image) are Base64-encoded as text encoded. Table 42–4 provides details.

Table 42–4 Capabilities

Capability	Description
Security	Not supported.
Filter/Transformation/Assign	Pass through.
Fanout	Supported.
Oracle B2B	Oracle B2B encodes the native payload to Base64 format. For this scenario, you must configure the Oracle B2B binding document definition handling to be opaque.

42.1.1.2 End-to-End Streaming with Attachments

This section describes use cases for end-to-end streaming of attachments.

42.1.1.2.1 SOAP with Attachments In this use case, the binary attachments (for instance an image) are passed end-to-end as a stream. Table 42–5 provides details.

Table 42–5 Capabilities

Capability	Description
Security	Not supported.
Filter/Transformation/Assign	Pass through. You must use an XPath extension function in Oracle BPEL Process Manager.
Fanout	Not supported.
Binding	WS binding creates stream iterators for the SOAP attachment.
Oracle BPEL Process Manager/Oracle Mediator	Oracle Mediator can perform a pass through without materializing it for synchronous routing rules (asynchronous routing rules are not supported). Oracle BPEL Process Manager persists it.
Tuning	Manage the database tablespace when using with Oracle BPEL Process Manager.

WSDL Code for defining SOAP with attachments	`<mime:part>`
	`  <mime:content part="bin" type="image/jpeg"/>`
	`</mime:part>`

Notes:

- You cannot stream attachments as part of a web service callback response.
- Deferred routing rules within Oracle Mediator do not support processing of attachments.
- The spring service component does not support processing MIME attachments. Only MTOM attachments are supported.
- You can use various binding components such as direct binding, web services, and so on to process large attachments. However, processing large attachments with direct binding is not recommended and results in out-of-memory errors.

Working with Streaming Attachments

Oracle Fusion Middleware web services enable you to pass large attachments as a stream. Unlike the JAX-RPC API, which treats attachments as if they are entirely in memory, streams make the programming model more efficient to use. Streams also enhance performance and scalability because there is no need to load the attachment into memory before service execution.

As with embedded attachments, streamed attachments conform to the multipart MIME binary format. On the wire, messages with streamed attachments are identical to any other SOAP message with attachments.

Example 42–1 provides a sample message with a streamed attachment. The first part in the message is the SOAP envelope (`<SOAP-ENV:Envelope...`). The second part is the attachment (for this example, `myImage.gif`).

Example 42–1 Sample Message with a Streamed Attachment

```
MIME-Version: 1.0
```

```
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
Content-Description: This is the optional message description.

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: NotSure/DoesntMatter

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
. . .
<DocumentName>MyImage.gif</DocumentName>
. . .
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/gif
Content-Transfer-Encoding: binary
Content-ID: AnythingYoudLike

...binary GIF image...
--MIME_boundary--
```

Creating Composites that Use MIME Attachments

Perform the following procedures to create composites that use MIME attachments.

To create composites that use MIME attachments:

1. Create a composite using a payload schema (for example, an inbound web service wired to an Oracle Mediator wired to an outbound web service).
2. Within the WSDL file of Oracle Mediator, perform the following steps:
   a. From the WSDL designer, open the Oracle Mediator WSDL file.
   b. Drag and drop bindings into the middle swimlane.
   c. Select the RPC binding.
   d. Enter a name.
   e. Go to **Source** view of the WSDL and modify the WSDL input and WSDL output with MIME multiparts.

```
<wsdl:input>
      <mime:multipartRelated>
          <mime:part>
              <soap:body parts="payload" use="literal"/>
          </mime:part>
          <mime:part>
              <mime:content part="bin" type="application/octet-stream"/>
          </mime:part>
      </mime:multipartRelated>
</wsdl:input>
```

   f. Add the MIME part in the request/response message.

```
<wsdl:message name="BPELProcess1RequestMessage">
    <wsdl:part name="payload" element="ns1:purchaseOrder" />
    <!--add below part-->
```

```
            <wsdl:part name="bin" type="xsd:base64Binary"/>
        </wsdl:message>
```

g. Add a namespace in the WSDL definitions.

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/">
```

When complete, the WSDL that references a MIME attachment is displayed.

```
<wsdl:definitions
  name="PhotoCatalogService"
  targetNamespace="http://examples.com/PhotoCatalog"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:types="http://examples.com/PhotoCatalog/types"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:tns="http://examples.com/PhotoCatalog">
  <wsdl:message name="addPhotoRequest">
    <wsdl:part name="photo" type="xsd:hexBinary"/>
  </wsdl:message>
  <wsdl:message name="addPhotoResponse">
    <wsdl:part name="status" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="replacePhotoRequest">
    <wsdl:part name="oldPhoto" type="xsd:string"/>
    <wsdl:part name="newPhoto" type="xsd:hexBinary"/>
  </wsdl:message>
  <wsdl:message name="replacePhotoResponse">
    <wsdl:part name="status" type="xsd:string"/>
  </wsdl:message>
  <wsdl:portType name="PhotoCatalog">
    <wsdl:operation name="addPhoto">
      <wsdl:input message="tns:addPhotoRequest"/>
      <wsdl:output message="tns:addPhotoResponse"/>
    </wsdl:operation>
    <wsdl:operation name="replacePhoto">
      <wsdl:input message="tns:replacePhotoRequest"/>
      <wsdl:output message="tns:replacePhotoResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="PhotoCatalogBinding" type="tns:PhotoCatalog">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="addPhoto">
      <wsdl:input>
        <mime:multipartRelated>
          <mime:part>
            <soap:body use="literal"/>
          </mime:part>
          <mime:part>
            <mime:content part="photo"
                          type="image/jpeg"/>
          </mime:part>
        </mime:multipartRelated>
      </wsdl:input>
      <wsdl:output>
        <mime:multipartRelated>
          <mime:part>
            <soap:body use="literal"/>
```

```
          </mime:part>
          <mime:part>
            <mime:content part="status" type="text/plain"/>
            <mime:content part="status" type="text/xml"/>
          </mime:part>
        </mime:multipartRelated>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="replacePhoto">
      <wsdl:input>
        <mime:multipartRelated>
          <mime:part>
            <soap:body parts="oldPhoto" use="literal"/>
          </mime:part>
          <mime:part>
            <mime:content part="newPhoto"
                          type="image/jpeg"/>
          </mime:part>
        </mime:multipartRelated>
      </wsdl:input>
      <wsdl:output>
            <soap:body parts="status" use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Performance Overhead and Pass Through Attachments

Because Oracle Mediator is stateless, there is no performance overhead with pass through attachments. However, Oracle BPEL Process Manager dehydrates attachments and has performance overhead, even for pass through attachments. Using Oracle BPEL Process Manager for attachments over a period of time, the SOA Infrastructure schema can grow to its maximum size and encounter memory issues. It is recommended that you extend the database tablespace appropriately for the SOA Infrastructure schema to accommodate large attachments. Simultaneously, you can use purge scripts to purge completed instances along with the attachments table. For information on purge scripts, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

In scenarios in which one BPEL process calls a second BPEL process within the same composite, the second BPEL process does not dehydrate the same attachment again.

In scenarios in which one BPEL process from composite 1 invokes a second BPEL process from composite 2 and optimization is disabled, composite 1 makes a SOAP call to composite 2. The second BPEL process does dehydrate attachments.

Properties for Streaming Attachments

To stream attachments, add the following properties in the `composite.xml` file. If optimization is enabled, then a native call is used instead of a SOAP call. Example 42–2 provides details.

Example 42–2 Properties for Streaming Attachments

```
<binding.ws
port="http://services.otn.com#wsdl.endpoint(MIMEService/MIMEService)"
xmlns:ns="http://xmlns.oracle.com/sca/1.0"
streamIncomingAttachments="true" streamOutgoingAttachments="true">
<!--Add this prop to reference bindings to make a SOAP call. -->
<property name="oracle.webservices.local.optimization">false</property>
```

```
</binding.ws>
```

For information about the `oracle.webservices.local.optimization` property, see "Managing SOA Composite Application Policies" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

> **Note:** Oracle Web Services Manager (OWSM) does not inspect or enforce policies on streamed attachments. For more information about OWSM, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

Reading and Encoding SOAP Attachment Content

The `ora:getAttachmentContent` function reads SOAP attachment content and encodes that data in Base64 format in a BPEL process by providing the BPEL variable as an argument, which has an `href` of the SOAP attachment. Example 42–3 shows how to use this function:

Example 42–3 ora:getAttachmentContent Function

```
<copy>
   <from expression="ora:getAttachmentContent('input','bin')"/>
   <to variable="initiateTaskInput" part="payload"
    query="/taskservice:initiateTask/task:task/task:attachment/task:content"/>
</copy>
```

Example 42–3 copies the attachment content, which has its `href` stored in the `"input/bin"` variable to the content variable, in Base64-encoded format.

Sending Attachment Streams

Oracle Mediator can pass an attachment stream to only one target receiver. The receiver can be another component or a web service/adapter. The second target cannot receive the attachment. Oracle BPEL Process Manager supports sending the attachment stream to multiple receivers. For Oracle BPEL Process Manager to send a stream to multiple receivers, it must read the attachment stream from the database using the `readBinaryFromFile` XPath function and pass the stream to the appropriate targets.

Sharing Attachments Using Synchronous Flows

When Oracle BPEL Process Manager-based composites share attachments using synchronous flows, it is necessary to use the same end-to-end transaction. This is applicable to composites that are colocated and use local/optimized calls. This can be achieved by setting the property shown in Example 42–4 on all the called BPEL components (callees) in the call chain.

Example 42–4 bpel.config.transaction Property

```
<property name="bpel.config.transaction" many="false"
type="xs:string">required</property>
```

If such composites do not execute as part of the same transaction context, the attachment data saved by the first BPEL component in the call chain is not visible to the other BPEL components in the call chain. In addition, they incur database locking and timeout exceptions:

```
"ORA-02049: timeout: distributed transaction waiting for lock"
```

42.1.1.2.2 Attachment Options of File/FTP Adapters In this use case, the adapter streams the binary data to a database store and publishes an `href` to the service engine (Oracle BPEL Process Manager or Oracle Mediator). Table 42–6 provides details.

Table 42–6 Capabilities

Capability	Description
Security	N/A.
Filter/Transformation/Assign	Filters and transformations on the attachment are not supported.
Fanout	Supported.
Binding	The adapter streams the non-XML to the database as a binary large object (BLOB) and passes the key to the service engines.
Oracle BPEL Process Manager/Oracle Mediator	Supported.
Tuning	▪ Extend the database tablespace for the Oracle SOA Suite schema. ▪ Delete the attachments after message processing completion.
Documentation	See *Oracle Fusion Middleware User's Guide for Technology Adapters*.

Writing Attachments Using an Outbound File Adapter

Example 42–5 shows a sample schema that can be used by the file adapter to write attachments to disk.

Example 42–5 Schema for Writing Attachments to Disk

```
<?xml version="1.0" encoding="windows-1252" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://xmlns.oracle.com/attachment"
            targetNamespace="http://xmlns.oracle.com/attachment"
            elementFormDefault="qualified">
            <xsd:element name="attach">
              <xsd:complexType>
                 <xsd:attribute name="href" type="xsd:string"/>
              </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Use Oracle Mediator in the flow to map the attachment part from the source (Oracle Mediator) to the target (file adapter) using an Oracle Mediator assign.

If you use Oracle BPEL Process Manager, the attachment is written to the dehydration store, which slows down the process.

Transforming Attachments with the ora:doStreamingTranslate XPath Function

Use of the `ora:doStreamingTranslate` XPath function is only recommended while transforming attachments within an Oracle BPEL Process Manager/Oracle Mediator composite. This function expects the attachment location to be a relative path on the server. This function cannot translate incoming attachment streams.

For more information about this function, see Section B.2.7, "doStreamingTranslate."

42.1.1.2.3 Oracle B2B Attachment In this use case, Oracle B2B stores the binary data to a database and publishes an `href` to the service engine (Oracle BPEL Process Manager or Oracle Mediator) based on an Oracle B2B-defined XSD. Oracle B2B protocols define the attachment. Table 42–7 provides details.

Table 42–7 Capabilities

Capability	Description
Security	N/A.
Filter/Transformation/Assign	Filters and transformations on the attachment are not supported.
Fanout	Supported.
Binding	Oracle B2B passes it as an `href` key to service engines.
Tuning	Extend the database tablespace for the Oracle SOA Suite schema.

42.1.1.3 Adding MTOM Attachments to Web Services

Within a SOA composite application, you must attach the Oracle WS-MTOM policy to service and reference binding components to receive and send MTOM (MIME binary) attachments within Oracle SOA Suite. When a service binding component (defined under `binding.ws` in the `composite.xml` file) is configured with an Oracle WS-MTOM policy, Oracle SOA Suite's MTOM message handling feature is used. When a reference binding component (also defined under `binding.ws` in the `composite.xml` file) is configured with an Oracle MTOM policy, Oracle SOA Suite sends MTOM-compliant messages with attachments.

Note the following issues with MTOM attachments.

- When attachments are inline and encoded, Oracle recommends that you not use the file adapter to write attachments to a file.
- The default `mtomThreshold` value is 1024 bytes and cannot be modified. If an attachment is less than 1024 bytes, for outbound configurations, Oracle SOA Suite sends it as an inline attachment. If the size is greater than 1024 bytes, then the attachment is sent as an attachment part with an `href` attribute in the message, and is sent as a WSDL-defined format on the wire. However, if the incoming request (for example, from a different web services provider) has an `xop href` node for small binary data (that is, size is less than 1024 bytes), Oracle SOA Suite uses the same `href` attribute in the payload in the flow trace. For example:

```
<xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
 href="cid:e29caf23dc8045908451fdfaafa26dce" />
```

- If service binding component of a composite does not include an Oracle WS-MTOM policy reference, this indicates that the service can accept non-MTOM messages. This indicates that the calling composite (the appropriate reference binding) does not have an Oracle WS-MTOM policy reference and can send out non-MTOM messages to that service.
- MTOM streaming of attachments is not supported by Oracle SOA Suite.
- MTOM attachments are supported only with web service bindings. Other bindings (for example, HTTP bindings) are not supported.
- Oracle Mediator pass through scenarios are supported. If Oracle Mediator does not contain any transformation or assign statements, it is known as a pass through

Oracle Mediator. The message and attachment received are propagated to the target without modifying the payload and attachment. Likewise, multiple MTOM attachments in the same message can be sent and received by Oracle SOA Suite.

- Oracle recommends that you not use both streaming and the MTOM message handling feature for sending and receiving attachments. Use either streaming or the MTOM message handling feature.

> **Note:** If the input is of type `text/xml`, there is no significant decrease in file size when sending files in MTOM format.

- As a best practice, Oracle recommends that you not use the XSLT Mapper to propagate binary data. Instead, use an assign activity. If you must use a style sheet to propagate binary data, it is recommended that you use the `xsl:copy-of` instruction (`copy-of` copies everything, including attributes) or use custom functions to copy attributes from source to target.
- MTOM attachments should not be gigabytes in size. Instead, use the SOAP with attachments streaming feature for very large attachments. For more information, see Section 42.1.1.2.1, "SOAP with Attachments."

42.1.1.3.1 Outbound Composite SOAP Messages Are Not Optimized If Only a WS-MTOM Policy is Used Unless a SOAP message passed to the `dispatch.invoke()` call of the SOA Infrastructure already contains `BinaryTextIml` nodes, binary data transmission is not optimized on the wire. Therefore, there is no guarantee that binary data optimization is always performed when a WS-MTOM policy is configured. The only way to ensure that optimization is performed is if the SOA MTOM configuration is also specified. The WS-MTOM policy guarantees the proper content-type setting with or without the SOA MTOM settings.

For example, assume you create a SOA composite application without `BinaryTextIml` nodes in the SOAP message that consists of the following components:

- A Java API for an XML Web Services (JAX-WS), MTOM-enabled, client service binding component
- An MTOM-enabled BPEL service component
- A JAX-WS web service reference binding component with MTOM

The JAX-WS, MTOM-enabled, client service binding component invokes the BPEL service component. The BPEL service component then invokes the JAX-WS, web service, reference binding component. The SOAP message from the JAX-WS client service binding component to the BPEL service component is MTOM-optimized. However, from the BPEL service component to the JAX-WS, web service reference binding component, the message is base64binary-enabled, and not MTOM-optimized.

42.1.1.4 Processing Large XML with Repeating Constructs

This section describes use cases for processing large XML with repeating constructs.

42.1.1.4.1 Debatching with the File/FTP Adapter In this use case, the inbound adapter splits a source document into multiple batches of records, each of which initiates a composite instance. Table 42–8 provides details.

Table 42–8 Capabilities

Capability	Description
Security	N/A.
Filter/Transformation/Assign	Supported.
Fanout	Supported.
Binding	The file/FTP adapter debatches it to a small chunk based on the native XSD (NXSD) definition.
Oracle BPEL Process Manager/Oracle Mediator	Supported.
Tuning	For repeating structures, XSLT is supported for scenarios in which the repeating structure is of smaller payloads compared to the overall payload size. Substitution with assign activities is preferred, as it performs a shadow copy.
Documentation	See *Oracle Fusion Middleware User's Guide for Technology Adapters*.

42.1.1.4.2 Chunking with the File/FTP Adapters In this use case, a loop within a BPEL process reads a chunk of records at a time and process (that is, cursor). Table 42–9 provides details.

Table 42–9 Capabilities

Capability	Description
Security	Supported.
Filter/Transformation/Assign	Supported.
Fanout	Supported.
Oracle BPEL Process Manager/Oracle Mediator	Supported only from Oracle BPEL Process Manager.
Documentation	See *Oracle Fusion Middleware User's Guide for Technology Adapters*.

42.1.1.5 Processing Large XML Documents with Complex Structures

This section describes use cases for processing very large XML documents with complex structures.

42.1.1.5.1 Streaming with the File/FTP Adapters In this use case, very large XML files are streamed through Oracle SOA Suite. Table 42–10 provides details.

Table 42–10 Capabilities

Capability	Description
Security	N/A.
Filter/Transformation/Assign	Supported, but must optimize to avoid issues.
Fanout	Supported.
Binding	The adapter streams the payload to a database as an SDOM and passes the key to the service engines.
Documentation	See *Oracle Fusion Middleware User's Guide for Technology Adapters*.

42.1.1.5.2 Oracle B2B Streaming In this use case, large XML files are passed by Oracle B2B to Oracle SOA Suite as an SDOM. This only occurs when a large payload size is defined in the Oracle B2B user interface. Table 42–11 provides details.

Table 42–11 Capabilities

Capability	Description
Security	N/A.
Filter/Transformation/Assign	Supported, but must optimize to avoid issues.
Fanout	Supported.
Binding	Oracle B2B streams the payload to a database as SDOM and passes the key to the service engines.
Oracle BPEL Process Manager/Oracle Mediator	Can use an XPath extension function to manipulate the payload.

42.1.2 Limitations on Concurrent Processing of Large Documents

This section describes the limitations on concurrent processing of large documents.

42.1.2.1 Opaque Schema for Processing Large Payloads

There is a limitation when you use an opaque schema for processing large payloads. The entire data for the opaque translator is converted to a single Base64-encoded string. An opaque schema is generally used for smaller data. For large data, use the attachments feature instead of the opaque translator.

42.1.3 General Tuning Recommendations

This section provides general tuning recommendations.

For more information about Oracle SOA Suite tuning and performance, see *Oracle Fusion Middleware Performance and Tuning Guide*.

42.1.3.1 General Recommendations

This section provides general tuning recommendations.

- Increase the JTA transaction timeout to `500` seconds in Oracle WebLogic Server Administration Console. For instructions, see section "Resolving Connection Timeouts" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.
- In Oracle Enterprise Manager Fusion Middleware Control, set the audit level to **Off** or **Production** at the SOA composite application level. See Section 42.1.3.2, "Setting Audit Levels from Oracle Enterprise Manager for Large Payload Processing" for additional information.
- Uncomment the following line in `setDomainEnv.sh` (for Linux) or `setDomainEnv.bat` (for Windows) for `JAVA_OPTIONS`, and restart the server. If this line does not exist, add it. Without this setting, large payload scenarios fail with `ResourceDisabledException` for the dehydration data source.

  ```
  -Dweblogic.resourcepool.max_test_wait_secs=30
  ```

- Update the heap size in `setSOADomainEnv.sh` or `setDomainEnv.bat` as follows:

  ```
  DEFAULT_MEM_ARGS="-Xms1024m -Xmx2048m"
  ```

- Use optimized translation functions, which are available while performing transformations and translations of large payloads (for example, `ora:doTranslateFromNative`, `ora:doTranslateToNative`, `ora:doStreamingTranslate`, and so on).

  For information about these functions, see Appendix B, "XPath Extension Functions."

- Extend data files for handling large attachments. For more information, see the *Oracle Database Administrator's Guide*.
- If you are processing large documents and run into timeout errors, perform the following tasks:
  - Increase the timeout property value.
  - Increase the **Stuck Thread Max Time** property value.

  Increase the timeout property value as follows:

  1. Log in to Oracle Web Services Manager Administration Console.
  2. Navigate to **Deployments** > **soa-infra** > **EJBs**.
  3. Click each of the following beans, select **Configuration**, and increase the timeout value:
     - **BpelEngineBean**
     - **BpelDeliveryBean**
     - **CompositeMetaDataServiceBean**

  Increase the **Stuck Thread Max Time** property value as follows:

  1. Follow the instructions in Chapter "Using the WebLogic 8.1 Thread Pool Model" of *Oracle Fusion Middleware Performance and Tuning for Oracle WebLogic Server*.

42.1.3.2 Setting Audit Levels from Oracle Enterprise Manager for Large Payload Processing

For large payload processing, turn off audit level logging for the specific composite. You can set the composite audit level option to **Off** or **Production** in Oracle Enterprise Manager Fusion Middleware Control. If you set the composite audit level option to **Development**, then it serializes the entire large payload into an in-memory string, which can lead to an out-of-memory error.

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

42.1.3.3 Using the Assign Activity in Oracle BPEL Process Manager/Oracle Mediator

When using the assign activity in Oracle BPEL Process Manager or Oracle Mediator to manipulate large payloads, do not assign the complete message. Instead, assign only the part of the payload that you need.

In addition, when using the assign activity in Oracle BPEL Process Manager, Oracle recommends using local variables instead of process variables, wherever possible. Local variables are limited to the scope of the BPEL process. These get deleted from memory and from the database after you close the scope. However, the life cycle of a global variable is tied with the instance life cycle. These variables stay in memory or

remain on disk until the instance completes. Thus, local variables are preferred to process or global variables.

42.1.3.4 Using XSLT Transformations on Large Payloads (For Oracle BPEL Process Manager)

Oracle recommends that you not perform XSLT transformations on large payloads using Oracle Mediator. Doing so results in out-of-memory errors when XSLT operations must traverse the entire document. Instead, use Oracle BPEL Process Manager.

Until 11*g* Release 1 11.1.1.3, for XSLT operations in Oracle BPEL Process Manager, the result was cached into memory as a whole document in binary XML format. For large document processing, this caused out-of-memory errors. Starting with 11*g* Release 1 11.1.1.4, a the `streamResultToTempFile` property has been added. This property enables XSLT results to be streamed to a temporary file and then loaded from the temporary file. Set `streamResultToTempFile` to `yes` when processing large payload using XSLT. The default value is `no`.

This property is applicable when using the following BPEL XPath functions:

- `ora:processXSLT('template','input','properties'?)`
- `ora:doXSLTransformForDoc ('template','input','name', 'value')`

To configure large XML documents to be processed using XSLT:

1. Create a BPEL common properties schema. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace ="http://schemas.oracle.com/service/bpel/common"
          xmlns:common = "http://schemas.oracle.com/service/bpel/common"
          xmlns:xs = "http://www.w3.org/2001/XMLSchema"
          elementFormDefault="qualified" blockDefault="#all">

  <xs:element name="serviceProperties"  type="common:PropertiesType"/>
  <xs:element name="anyProperties"  type="common:ArrayOfNameAnyTypePairType"/>
     <xs:complexType name="NameValuePairType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="value" type="xs:string"/>
    </xs:sequence>
   </xs:complexType>

   <xs:complexType name="ArrayOfNameValuePairType">
    <xs:sequence>
      <xs:element name="item" type="common:NameValuePairType"
 maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>

   <xs:complexType name="NameAnyTypePairType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="value" type="xs:anyType"/>
    </xs:sequence>
   </xs:complexType>

   <xs:complexType name="ArrayOfNameAnyTypePairType">
    <xs:sequence>
      <xs:element name="item" type="common:NameAnyTypePairType"
```

```
 maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>

   <xs:complexType name="PropertiesType">
    <xs:sequence>
      <xs:element name="property" type="common:NameValuePairType"
 maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>

   <xs:complexType name="ArrayOfAnyTypeType">
    <xs:sequence>
      <xs:element name="item" type="xs:anyType" maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
</xs:schema>
```

2. Within a BPEL process, add the namespace in the `import` section:

```
xmlns:common = "http://schemas.oracle.com/service/bpel/common"
```

3. Create a global variable (for this example, named `propertiesXMLVar`):

```
 <variable name="propertiesXMLVar" element="common:anyProperties"/>
```

4. Set the `streamResultToTempFile` property to `yes`. This assign activity should exist before using performing an XSLT transformation.

```
<assign name="Assign_xsltprop">
      <copy>
        <from>
           <common:anyProperties>
             <common:item>
                   <common:name>streamResultToTempFile</common:name>
                   <common:value>yes</common:value>
             </common:item>
           </common:anyProperties>
        </from>
        <to variable="propertiesXMLVar"/>
      </copy>
</assign>
```

42.1.3.5 Using XSLT Transformations for Repeating Structures

In scenarios in which the repeating structure is of smaller payloads compared to the overall payload size, Oracle recommends using XSLT transformation because the current XSLT implementation materializes the entire DOM in memory. For example, use `PurchaseOrder.LineItem.Supplier` (a subpart of a large payload).

You can also substitute it with the assign activity, as it performs a shadow copy. Although a shadow copy does not materialize DOM, it creates a shadow node to point to the source document.

You can also use the following optimized translation functions while performing transformations/translations of large payloads:

- `ora:doTranslateFromNative`
- `ora:doTranslateToNative`
- `ora:doStreamingTranslate`

For more information about the usage of these functions, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

42.1.3.6 Processing Large Documents in Oracle B2B

For processing large documents in Oracle B2B, tune the following parameters:

- `mdsCache`
- `Cache Size`
- `Protocol Message Size`
- `Number of threads`
- `Stuck Thread Max Time`
- `Tablespace`

The following sections describe the parameters you must set for processing large documents in Oracle B2B:

42.1.3.6.1 MDSInstance Cache Size To set Metadata Service (MDS) instance cache size, the property and value must be added to the `$DOMAIN_HOME/config/soa-infra/configuration/b2b-config.xml` file, as shown in Example 42–6.

Example 42–6 MDSInstance Cache Size

```
<property>
 <name>b2b.mdsCache</name>
 <value>200000</value>
 <comment>MDS Instance cache size </comment>
</property>
```

42.1.3.6.2 Protocol Message Size If Oracle B2B wants to send or receive more than 10 MB of message or the import/export configuration is more than 10 MB, then change the following setting accordingly at the Oracle WebLogic Server Administration Console:

1. In the **Domain Structure**, select **Environment** > **Servers**.
2. In the **Name** column of the table, select ***soa_server***.
3. Select the **Protocols** tab.
4. Change the value for **Maximum Message Size**.

This setting can also be added/modified in the `$DOMAIN_HOME/config/config.xml` file next to the server name configuration, as shown in Example 42–7.

Example 42–7 max-message-size Property

```
<name>soa_server1</name>
<max-message-size>150000000</max-message-size>
```

> **Note:** By default, `max-message-size` is not available in the `config.xml` file.

42.1.3.6.3 Number of Threads This parameter helps to improve the message processing capability of Oracle B2B and must be set in the `$DOMAIN_`

HOME/config/soa-infra/configuration/b2b-config.xml file. Example 42–8 provides an example.

Example 42–8 Number of Threads

```
<property>
  <name>b2b.inboundProcess.threadCount</name>
  <value>5</value>
  <comment></comment>
</property>
<property>
  <name>b2b.inboundProcess.sleepTime</name>
  <value>10</value>
  <comment></comment>
</property>
<property>
  <name>b2b.outboundProcess.threadCount</name>
  <value>5</value>
  <comment></comment>
</property>
<property>
  <name>b2b.outboundProcess.sleepTime</name>
  <value>10</value>
  <comment></comment>
</property>
<property>
  <name>b2b.defaultProcess.threadCount</name>
  <value>5</value>
  <comment></comment>
</property>
<property>
  <name>b2b.defaultProcess.sleepTime</name>
  <value>10</value>
  <comment></comment>
</property>
```

42.1.3.6.4 Stuck Thread Max Time The **Stuck Thread Max Time** parameter checks the number of seconds that a thread must be continually working before the server considers the thread stuck. You must change the following setting in the Oracle WebLogic Server Administration Console:

1. In the **Domain Structure**, select **Environment** > **Servers**.
2. In the **Name** column of the table, select ***soa_server***.
3. Select the **Tuning** tab.
4. Change the value for **Stuck Thread Max Time**.

42.1.3.6.5 Tablespace If you must store more than a 150 MB configuration in the data file, then you must extend or add the data file to increase the tablespace size, as shown in Example 42–9.

Example 42–9 Extension of Data File

```
ALTER TABLESPACE sh_mds add DATAFILE 'sh_mds01.DBF' SIZE 100M autoextend on next
 10M maxsize unlimited;
ALTER TABLESPACE sh_ias_temp add TEMPFILE 'sh_ias_temp01.DBF' SIZE 100M autoextend
 on next 10M maxsize unlimited;
```

42.1.3.7 Using XPath Functions to Write Large XSLT/XQuery Output to a File System

You can use the following functions to write the results of large XSLT/XQuery operations to a temp file in a directory system. The document is then loaded from the temp file when needed. This eliminates the need for caching an entire document as binary XML in memory.

- `ora:processXSLT`
- `ora:doXSLTransformForDoc`

With the `ora:processXSLT` function, you use the `properties` argument to enable this functionality.

```
ora:processXSLT('template','input','properties'?)
```

You retrieve the value of this argument within your XSLT in a way similar to extracting data from XSL variables. The `properties` argument is an XML element of the structure shown in Example 42–10. For large payload results (for example, above 10 MB), set `streamResultToTempFile` to `yes`. For small payload results in which you do not need to write results to a temp file, leave this property set to its default value of `no`.

Example 42–10 properties XML

```
<propertiesXMLVar>
  <common:item  xmlns:common="http://schemas.oracle.com/service/bpel/common">
    <common:name>streamResultToTempFile</common:name>
    <common:value>yes</common:value>
  </common:item>
</propertiesXMLVar>
```

Within the XSLT, the parameters are accessible through the name of `streamResultToTempFile` and its value of `yes`.

In Oracle BPEL Process Manager, a literal assign is performed to populate the properties for `ora:processXSLT('template','input','properties'?)`.

For more information on using this function, see Section B.2.53, "processXSLT."

With the `ora:doXSLTransformForDoc` function, you set the `name` and `value` properties to enable this functionality.

```
ora:doXSLTransformForDoc ('template','input','name', 'value')
```

With this function, the `name` of `streamResultToTempFile` and the `value` of `yes` are passed.

For more information on using the function, see Section B.2.11, "doXSLTransformForDoc."

42.2 Best Practices for Handling Large Metadata

This section provides recommendations for handling large metadata.

42.2.1 Boundary on the Processing of Large Numbers of Activities in a BPEL Process

There is a limit to the number of activities that can be executed in a BPEL process. When you exceed this limit, system memory fills up, which can cause timeouts to

occur. For example, with the following parameters, two fault instances occur due to a timeout:

- 100 threads
- 1 second of think time
- 1000 incoming request messages

Try to keep the number of incoming request messages at a proper level to ensure system memory stability.

42.2.2 Using Large Numbers of Activities in BPEL Processes (Without FlowN)

To deploy BPEL processes that have a large number of activities (for example, 50,000), the following settings are required:

```
MEM_ARGS: -Xms512m -Xmx1024m -XX:PermSize = 128m -XX:MaxPermSize
= 256m

Number of Concurrent Threads = 20

Number of Loops = 5 Delay = 100 ms
```

The above settings enable you to deploy and execute BPEL processes, which use only while loops without the flowN activities, successfully.

42.2.3 Using Large Numbers of Activities in BPEL Processes (With FlowN)

To deploy BPEL processes that have a large number of activities (for example, 50,000), the following settings are required:

```
USER_MEM_ARGS: -Xms2048m -Xmx2048m -XX:PermSize=128m
-XX:MaxPermSize=256m

Number of Concurrent Threads= 10

Number of Loops=5 Delay=100 ms
```

Set the **StatsLastN** property to `-1` in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control.

The above settings enable you to deploy and execute BPEL processes, which use the flowN activities, successfully.

For more information, see Chapter 9, "Using Parallel Flow in a BPEL Process."

42.2.4 Using a Flow With Multiple Sequences

BPEL processes that have up to 7000 activities can be deployed and executed successfully with the following settings:

```
USER_MEM_ARGS: -Xms2048m -Xmx2048m -XX:PermSize=128m
-XX:MaxPermSize=256m
```

> **Note:** If you deploy BPEL processes with more than 8000 activities, Oracle BPEL Process Manager compilation throws errors.

42.2.5 Using a Flow with One Sequence

BPEL processes that have up to 7000 activities can be deployed and executed successfully with the following settings:

```
USER_MEM_ARGS: -Xms2048m -Xmx2048m -XX:PermSize=128m
-XX:MaxPermSize=512m
```

Note: If you deploy BPEL processes with more than 10,000 activities, the Oracle BPEL Process Manager compilation fails.

42.2.6 Using a Flow with No Sequence

You can deploy and execute BPEL processes that have a large number of activities (for example, up to 5000) successfully.

There is a probability that the BPEL compilation may fail for 6000 activities.

42.2.7 Large Numbers of Oracle Mediators in a Composite

Oracle recommends that you not have more than 50 Oracle Mediators in a single composite. Increase the JTA Transaction timeout to a high value based on the environment.

42.2.8 Importing Large Data Sets in Oracle B2B

Oracle recommends that you do not use browsers for large data set imports, and that you use the command-line utility. The following utility commands are recommended for large data configuration:

- `purge`: This command is used to purge the entire repository.
- `import`: This command is used to import the specified ZIP file.
- `deploy`: This command is used to deploy an agreement with whichever name is specified. If no name is specified, then all the agreements are deployed.

However, the `purgeimportdeploy` option is not recommended for transferring or deploying the Oracle B2B configuration.

For more information, see *Oracle Fusion Middleware User's Guide for Oracle B2B*.

42.3 Best Practices for Handling Large Numbers of Instances

This section provides recommendations for handling large numbers of instance and fault metrics.

42.3.1 Instance and Rejected Message Deletion with the Purge Script

Deleting thousands of instances and rejected messages in Oracle Enterprise Manager Fusion Middleware Control takes time and can result in a transaction timeout. If you must perform this task, use the PL/SQL purge script for instance and rejected message deletion.

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

42.3.2 Improving the Loading of Pages in Oracle Enterprise Manager Fusion Middleware Control

You can improve the loading of pages that display large amounts of instance and fault data in Oracle Enterprise Manager Fusion Middleware Control by setting two

properties in the **Display Data Counts** section of the SOA Infrastructure Common Properties page.

These two properties enable you to perform the following:

- Disable the fetching of instance and fault count data to improve loading times for the following pages:
  - Dashboard pages of the SOA Infrastructure, SOA composite applications, service engines, and service components
  - Delete with Options: Instances dialog

  These settings disable the loading of all metrics information upon page load. For example, on the Dashboard page for the SOA Infrastructure, the values that typically appear in the **Running** and **Total** fields in the **Recent Composite Instances** section and the **Instances** column of the **Deployed Composites** section are replaced with links. When these values are large, it can take time to load this page and other pages with similar information.
- Specify a default time period that is used as part of the search criteria for retrieving recent instances and faults for display on the following pages:
  - Dashboard pages and Instances pages of the SOA Infrastructure, SOA composite applications, service engines, and service components
  - Dashboard pages of services and references
  - Faults and Rejected Messages pages of the SOA Infrastructure, SOA composite applications, services, and references
  - Faults pages of service engines and service components

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

43

Customizing SOA Composite Applications

This chapter describes how to customize SOA composite applications with the customization feature available with a BPEL process service component.

This chapter includes the following sections:

43.1 Introduction to Customizing SOA Composite Applications

This section describes the life cycle for customizing SOA composite applications. For example, assume the following organizations require use of the same composite, but with slight modifications:

- A core applications development team
- A vertical applications team
- A customer

The core applications development team creates a base customizable composite and delivers it to a vertical applications team that customizes it for a certain industry (for example, telecommunications). The tailored solution is then sold to a telecommunications customer that further customizes the composite for their specific geographic business needs. Essentially, there is a base composite and several layers of customized composites. At a later time in the composite life cycle, the core applications development team creates the next version of the base composite, triggering an upgrade cycle for the vertical applications team and the customer.

Note: Do *not* customize the same SOA composite application for different layer values. Only a single layer value for customization is supported. If you must support another layer value, always import the base composite into a different project and change the composite name to be specific to the layer value you want to customize. This approach is also useful for deployments in which you do not want to deploy different layer values with the same composite name.

43.2 Creating the Customizable Composite

This section provides an overview of the steps required for creating the customizable, base SOA composite application.

43.2.1 How to Create the Customizable Composite

To create the customizable composite:

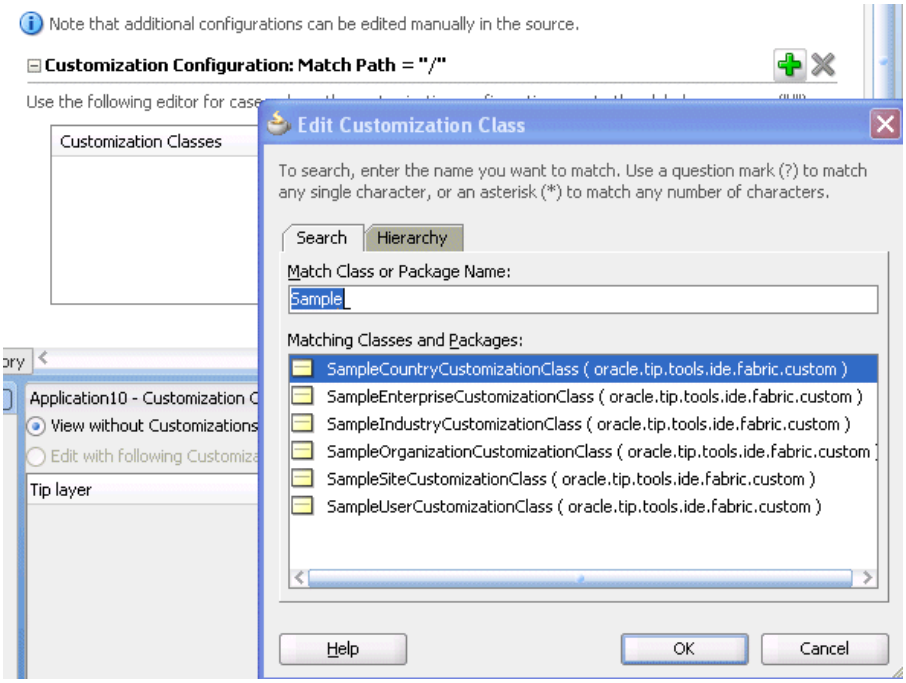
1. Start Oracle JDeveloper and select the **Default Role**.
2. From the **File** menu, select **New** > **Applications** > **SOA Application**, and click **OK**.
3. Follow the steps in the Create SOA Application wizard.
4. On the Create SOA Application dialog, select both **Composite With BPEL Process** and the **Customizable** checkbox.
5. Design the BPEL process.
6. Customize the BPEL process by creating a scope activity. This action is required because by default the BPEL process is not customizable.

   Note: You can only customize the `composite.xml` file, `.bpel` file (for Oracle BPEL Process Manager), `.mplan` file (for Oracle Mediator), and `.componentType` file when using Oracle JDeveloper in the Customization Developer role.

7. Right-click the scope and select **Customizable**.
8. In the Application Navigator, expand **Application Resources** > **Descriptors** > **ADF META_INF**.
9. Open the **adf-config.xml** file and select the **MDS Configuration** tab.
10. Click the **Add** icon to add the required customization classes, as shown in Figure 43–1.

    In real environments, the customization classes are provided by the core applications team. When you use your own customization classes, you must add your customization class JAR file to your project to make the classes available for the **adf-config.xml** file.

Figure 43–1 Customization Classes



11. Right-click the SOA project and select **Deploy**. This creates a JAR file package. This JAR is also known as a SOA archive (SAR).
12. Check the application into a source code control system. The file is now ready for delivery to the vertical applications team.

For information on how to write customization classes, see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

43.2.2 How to Create Customization Classes

This section describes how to create customization classes. In this example, you create a class for a customization layer named `MyCustomizationLayer`.

To create customization classes:

1. Invoke the Create Java Class Wizard in Oracle JDeveloper by selecting **File** > **New** > **General** > **Java** > **Java Class**. If this selection does not appear, ensure that the **All Technologies** tab is selected at the top of the New Gallery dialog.
2. Create a Java class extending from the following class:

   ```
   oracle.tip.tools.ide.fabric.custom.GenericSOACustomizationClass
   ```

3. Provide the following content for the customization class.

   ```
   package myCustomizationPackage;

   import oracle.tip.tools.ide.fabric.custom.GenericSOACustomizationClass;

   public class MyCustomizationClass extends GenericSOACustomizationClass {

       public MyCustomizationClass() {
           super();

           // set the customization layer name
   ```

```
        setName("MyCustomizationLayer");
    }
}
```

Note that the Create Java Class Wizard automatically generates the following content:

```
package myCustomizationPackage;

import oracle.tip.tools.ide.fabric.custom.GenericSOACustomizationClass;

public class MyCustomizationClass extends GenericSOACustomizationClass {
    public MyCustomizationClass(String string, String string1) {
        super(string, string1);
    }

    public MyCustomizationClass() {
        super();
    }
}
```

4. For the customization class to have the correct customization layer, set the customization layer name by adding the following to the constructor without parameters.

```
 // set the customization layer name
        setName("MyCustomizationLayer");
```

You can also optionally remove the constructor with parameters.

5. To make the customization class effective, compile the customization class by building/making the SOA project.
6. Ensure that the build succeeds.

43.2.3 How to Add XSD or WSDL Files

You can add an XML schema or WSDL document in Oracle JDeveloper when logged in with the **Customization Developer** role.

1. Right-click the Oracle SOA Suite project in the Application Navigator.
2. Select **SOA**.
3. Select the artifact to create:
   - **Create XML Schema**

     Invokes the Create XML Schema dialog for adding a new XML schema file in the project. When complete, the new schema file automatically opens.
   - **Create WSDL Document**

     Invokes the Create WSDL dialog to add a new WSDL file in the project.

43.2.4 How to Search for Customized Activities in a BPEL Process

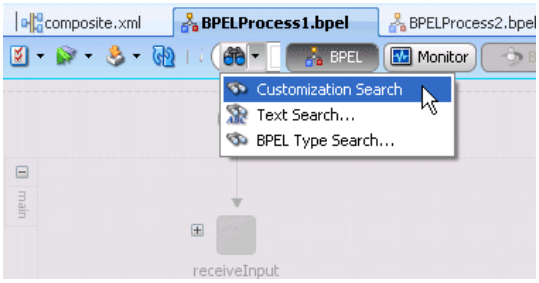
You can search for customized activities in a BPEL process in Oracle JDeveloper.

To search for customized activities:

1. Access Oracle JDeveloper using the **Customization Developer** role.

2. In the **Search** menu for the BPEL process at the top of the designer, select **Customization Search**, as shown in Figure 43–2.

Figure 43–2 Customization Search Option



The search results display in the **Search for Customizations** tab of the Log window at the bottom of the designer.

43.2.5 What You May Need to Know About Editing Artifacts in a Customized Composite

The source of any artifact in Oracle JDeveloper (except for new artifacts created in the **Customization Developer** role) is editable in the **Customization Developer** role of another application.

For example:

1. Create a SOA composite application with the **Customization Developer** option selected.
2. Edit a `composite.xml` property in the composite (as an example, add the `passThroughHeader` property for an Oracle Mediator service component included in the composite).
3. Deploy the composite to a SAR file, and import the SAR file into another new composite.
4. Restart Oracle JDeveloper and open the new composite using the **Customization Developer** role.
5. Go to the **Source** view of the `composite.xml` file, and from the Property Inspector attempt to modify the `passThroughHeader` property value.

   Note that the property is editable.

43.2.6 What You May Need to Know About Resolving Validation Errors in Oracle JDeveloper

In the customization role, the metadata repository (MDS) merges customizations with the base metadata. The merging can result in an invalid XML document against its schema. MDS merging does not invoke a schema validation to ensure that the merging always creates a valid XML document. This can cause a problem for MDS clients that rely on the validity of the metadata to render their metadata UI editors.

Whenever a SOA file such as `composite.xml` becomes invalid, you must switch to **Source** view in Oracle JDeveloper to directly fix the XML source. If **Source** view is not editable (for example, you have accessed Oracle JDeveloper using the **Customization Developer** role), you must use the Structure window in Oracle JDeveloper to fix the XML source.

For example, assume you created a base SOA composite application with a BPEL process that included a customized scope. The SAR file for the base application was then imported into a new application in which the following components were added when accessing Oracle JDeveloper with the **Customization Developer** role:

- An outbound file adapter
- An invoke activity (added to the customizable scope) for invoking the file adapter

When version two of the base SOA composite application was created, a synchronous Oracle Mediator service component was added, which caused the routing rules to the BPEL process service component to be updated.

The SAR file for version two of the base application was then imported into the customized application. When the user accessed Oracle JDeveloper with the **Customization Developer** role, an invalid composite error was displayed. The `composite.xml` file in the Structure window showed the following invalid structure for the sequence of service components and reference binding components.

```
<component> </component>
<reference> </reference>
<component> </component>
```

The `<reference>` component (in this case, the outbound file adapter added when the user accessed Oracle JDeveloper with the **Customization Developer** role in version one of the base application) should have displayed last.

```
<component> </component>
<component> </component>
<reference> </reference>
```

To resolve this error, go to the Structure window and copy and paste these components into the correct order. This action resolves the composite validation error.

43.2.7 What You May Need to Know About Resolving a Sequence Conflict

Assume you perform the following steps.

1. Customize version 1 of a SOA composite application.

   For example, while using Oracle JDeveloper in the **Customization Developer** role, you add new activities into a customizable scope activity of the BPEL process. The BPEL process creates a sequence activity into which the new activities are added.

2. Create version 2 of the SOA composite application.

   In the version 2 composite, if new activities are added into the same customizable scope, a new sequence activity is created.

3. Import version 2 of the SOA composite application into a customized application.
4. Open Oracle JDeveloper in **Customization Developer** role.

   The following error is displayed:

   ```
   Sequence element is not expected
   ```

Perform the following steps to resolve the conflict:

1. Go to the Structure window.
2. Expand the sequence.
3. Copy each component and paste it into another sequence.

4. Delete the components in the sequence from which they were copied.
5. Delete the sequence when it is empty.

43.2.8 What You May Need to Know About Compiling and Deploying a Customized Application

When you deploy or compile a customized application at the core application, vertical application, or customer level, warning messages describing unexpected ID attributes are displayed, as shown in Example 43–1. You can safely ignore these messages. These messages display because the schema definition does not include these simple-type elements, which is expected behavior. These messages do not prevent the customized composite from being successfully deployed.

Example 43–1 Deployment or Compilation Errors

```
[scac] warning: in
 /scratch/qizhong/my-jdev/mywork/CompositeTestApp2/Project2/composite.xml(22,32):
 Schema validation failed for
 /scratch/qizhong/my-jdev/mywork/CompositeTestApp2/Project2/composite.xml<Line 22,
 Column 32>: XML-24535: (Error) Attribute
 'http://www.w3.org/XML/1998/namespace:id' not expected.
    [scac] warning: in
 /scratch/qizhong/my-jdev/mywork/CompositeTestApp2/Project2/composite.xml(23,32):
 Schema validation failed for
 /scratch/qizhong/my-jdev/mywork/CompositeTestApp2/Project2/composite.xml<Line 23,
 Column 32>: XML-24535: (Error) Attribute
 'http://www.w3.org/XML/1998/namespace:id' not expected.
```

43.3 Customizing the Vertical Application

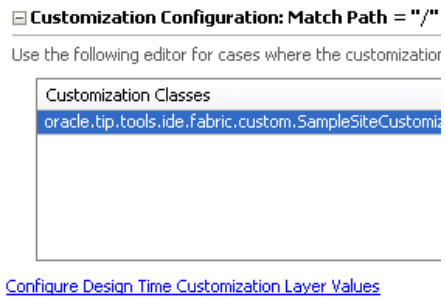
This section provides an overview of the steps required for customizing the vertical SOA composite application.

43.3.1 How to Customize the Vertical Application

To customize the vertical application:

1. Add the layer values for the customization layers through either of the following methods:
   a. To add application-specific layer values, click the **Configure Design Time Customization Layer Values** link, as shown in Figure 43–3.

Figure 43–3 Configure Design Time Customization Layer Values Link



   b. Add the layer values.

After you specify the values and save the file, the **CustomizationLayerValues.xml** file is displayed in the **MDS DT** folder under Application Resources. You can double-click the file in this location to open an editor for making additional modifications.

or

a. To add global values applicable to all applications, open the `CustomizationLayerValues.xml` file in `$JDEV_HOME/jdeveloper/jdev` and add the layer values for the customization layers. For example, add the value `Communications` to the `industry` layer.

```
<cust-layers xmlns="http://xmlns.oracle.com/mds/dt">
  <cust-layer name="industry">
    <cust-layer-value value="communications"
display-name="Communications"/>
  </cust-layer>
</cust-layers>
```

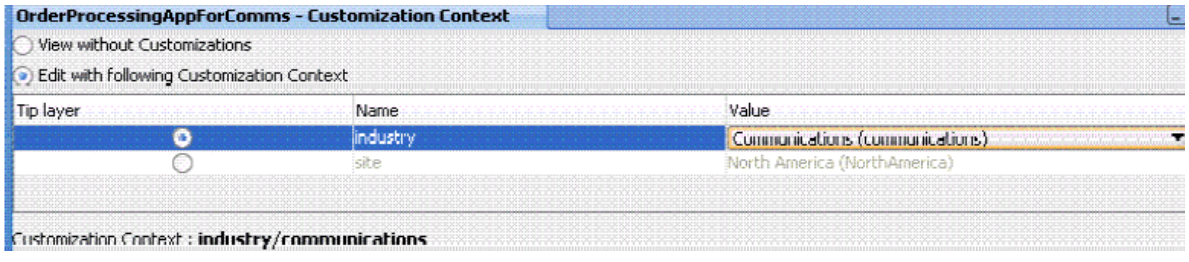
2. Start Oracle JDeveloper and select the **Default Role**.
3. Create a new SOA application with a different name than the core application.
4. From the **File** menu, select **Import** > **SOA Archive Into SOA Project**.
5. Click **Browse** to select the composite archive JAR file created by the core application team in Section 43.2, "Creating the Customizable Composite."
6. In the **Composite Name** field, enter a different name than the core SOA project.

> **Note:** Do not select any SOA project. You must create a new SOA project for the JAR file that you import.

7. Select the **Import for Customization** checkbox.
8. From the **Tools** menu, select **Preferences** -> **Roles** > **Customization Developer**.
9. Restart Oracle JDeveloper.

   The Customization Context dialog displays the available customization layers and layer values.
10. Select a layer and value to customize, as shown in Figure 43–4 (for this example, layer **industry** and value **Communications** are selected).

Figure 43–4 Customization Context

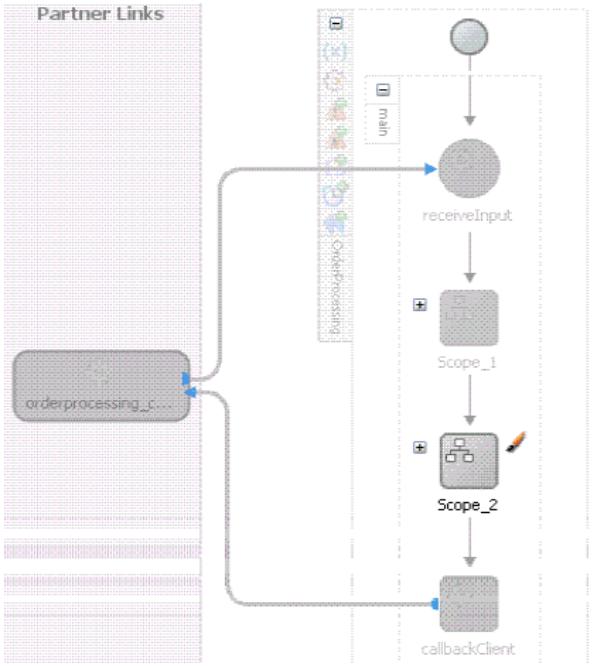


11. In the SOA Composite Editor, double-click the BPEL process to access Oracle BPEL Designer.

    You can only edit scope activities that have been set to customizable. In the example shown in Figure 43–5, the core applications team set only one scope to be

customizable. The other activities in the BPEL process are disabled and cannot be edited.

Figure 43–5 One Customizable Scope



12. Right-click the SOA project in the Application Navigator and select **Deploy** to create a JAR file of the customized composite (SAR).

    Since deployment is invoked with the customization role enabled, the base composite with the appropriate layers based on the current customization context is automatically merged.

13. Check in the application to a source code control system.

    The JAR file contains a merged composite that in turn acts as a base process for the next level of customization. The JAR file can now be delivered to the customer.

Note: You can create WSDL and XSD files while running Oracle JDeveloper in the **Customization Developer** role. In the Application Navigator, right-click the project name and select **SOA** > **Create WSDL** or **SOA** > **Create XSD**.

43.4 Customizing the Customer Version

This section provides an overview of the steps required for customizing the customer version of the SOA composite application.

43.4.1 How to Customize the Customer Version

How to customize the customer version:

1. Add the layer values for the customization layers through either of the following methods:

a. To add application-specific layer values, click the **Configure Design Time Customization Layer Values** link, as shown in Step 1 of Section 43.3, "Customizing the Vertical Application."

b. Add the layer values.

   After you specify the values and save the file, the **CustomizationLayerValues.xml** file is displayed in the **MDS DT** folder under Application Resources. You can double-click the file in this location to open an editor for making additional modifications.

or

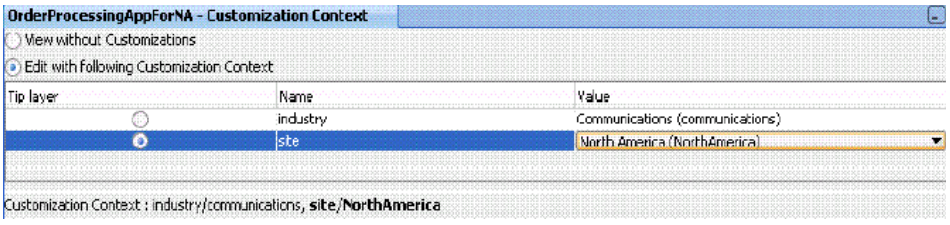
a. To add global values applicable to all applications, open the `CustomizationLayerValues.xml` file in `$JDEV_HOME/jdeveloper/jdev` and add the layer values for the customization layers. For example, add the values `North America` and `Asia Pacific` to the `site` layer.

```
<cust-layers xmlns="http://xmlns.oracle.com/mds/dt">
  <cust-layer name="site">
    <cust-layer-value value="communications" display-name="North America"/>
    <cust-layer-value value="communications" display-name="Asia Pacific"/>
  </cust-layer>
</cust-layers>
```

2. Start Oracle JDeveloper and select the **Default Role**.
3. Create a new SOA application with a different name than the core application or customized application.
4. From the **File** menu, select **Import** > **SOA Archive Into SOA Project**.
5. Click **Browse** to select the composite archive JAR file created by the vertical applications team in Section 43.3, "Customizing the Vertical Application."
6. Select the **Import for Customization** checkbox.
7. From the **Tools** menu, select **Preferences** -> **Roles** > **Customization Developer**.
8. Restart Oracle JDeveloper.

   The Customization Context dialog displays the available customization layers and layer values.
9. Select a layer and value to customize, as shown in Figure 43–6 (for this example, layer **site** and value **North America** are selected).

Figure 43–6 Customization Context



10. Customize the BPEL process.
11. Right-click the SOA project and select **Deploy** to create a JAR file (SAR) for the North American region.
12. Check the application into a source code control system.

43.5 Upgrading the Composite

This section provides an overview of the steps required for upgrading the SOA composite application to the next version.

43.5.1 How to Upgrade the Core Application Team Composite

The core application team fixes bugs, makes product enhancements. and creates the next version of the composite.

To upgrade the core application team composite:

1. Check out the application created in Section 43.2, "Creating the Customizable Composite" from source control.
2. Start Oracle JDeveloper and select the **Default Role**.
3. Make bug fixes and product enhancements.
4. Deploy the composite to create the next revision of the JAR file.
5. Deliver the JAR file to the vertical applications team.

43.5.2 How to Upgrade the Vertical Application Team Composite

The vertical applications team customizes the new base composite to create a version of the JAR file.

To upgrade the vertical application team composite:

1. Check out the application created in Section 43.3, "Customizing the Vertical Application" from source control.
2. Start Oracle JDeveloper and select the **Default Role**.
3. Open the checked-out application.
4. Select the project node in the Application Navigator to set the current project context. This is important because the import command in the next step imports the SOA archive into the selected project to upgrade the base.
5. From the **File** menu in Oracle JDeveloper, import the new revision of the JAR file created in Section 43.5.1, "How to Upgrade the Core Application Team Composite."
6. From the **Tools** menu, select **Preferences** -> **Roles** > **Customization Developer**.
7. Restart Oracle JDeveloper.
8. In the Customization Context dialog, select a layer and value to customize (for example, select layer **industry** and value **Communications**).
9. Open the BPEL process to see if the new base composite can be merged with layers for the above selected context.
10. Review the log window for potential warnings and errors.
11. If required, fix errors and warnings by modifying the process. This edits the layers behind the scenes.
12. Deploy the composite to create the next revision of the customized JAR file and deliver it to the customer for an upgrade.

43.5.3 How to Upgrade the Customer Composite

The customer follows the same procedures as the vertical applications team in Section 43.5.2, "How to Upgrade the Vertical Application Team Composite" to apply their layers to the new base composite.

44

Working with Domain Value Maps

This chapter describes how to use domain value maps to map the vocabulary used by different domains.

This chapter includes the following sections:

44.1 Introduction to Domain Value Maps

Domain value maps operate on actual data values that transit through the infrastructure at runtime. They enable you to map from one vocabulary used in a given domain to another vocabulary used in a different domain. For example, one domain may represent a city with a long name (`Boston`), while another domain may represent a city with a short name (`BO`). In such cases, you can directly map the values by using domain value maps. A direct mapping of values between two or more domains is known as point-to-point mapping. Table 44–1 shows a point-to-point mapping for cities between two domains:

Table 44–1 Point-to-Point Mapping

CityCode	CityName
BELG_MN_STLouis	BelgradeStLouis
BELG_NC	BelgradeNorthCarolina
BO	Boston
NP	Northport
KN_USA	KensingtonUSA
KN_CAN	KensingtonCanada

Each domain value map typically holds a specific category of mappings among multiple applications. For example, one domain value map may hold mappings for city codes and another may hold mappings for state codes.

Domain value map values are static. You specify the domain value map values at design time using Oracle JDeveloper, and then at runtime, the domain value map columns are looked up for values.

For information about editing domain value maps at runtime with Oracle SOA Composer, see Chapter 45, "Using Oracle SOA Composer with Domain Value Maps."

Note: To dynamically integrate values between applications, you can use the cross referencing feature of Oracle SOA Suite. For information about cross references, see Chapter 46, "Working with Cross References."

44.1.1 Domain Value Map Features

This section describes domain value map functionality.

44.1.1.1 Qualifier Support

Qualifiers qualify mappings. A mapping may not be valid unless qualified with additional information. For example, a domain value map containing a city code-to-city name mapping may have multiple mappings from `KN` to `Kensington` because `Kensington` is a city in both `Canada` and the `USA`. Therefore, this mapping requires a qualifier (`USA` or `Canada`) to qualify when the mapping becomes valid, as shown in Table 44–2.

Table 44–2 Qualifier Support Example

Country (Qualifier)	CityCode	CityName
USA	BO	Boston
USA	BELG_NC	Belgrade
USA	BELG_MN_Streams	Belgrade
USA	NP	Northport
USA	KN	Kensington
Canada	KN	Kensington

You can also specify multiple qualifiers for a domain value map. For example, as shown in Table 44–3, `BELG` to `Belgrade` mapping can also be qualified with a state name.

Table 44–3 Multiple Qualifier Support Example

Country (Qualifier)	State (Qualifier)	CityCode	CityName
USA	Massachusetts	BO	Boston
USA	North Carolina	BELG	Belgrade
USA	Minnesota	BELG	Belgrade
USA	Alabama	NP	Northport
USA	Kansas	KN	Kensington
Canada	Prince Edward Island	KN	Kensington

Qualifiers are used only to qualify the mappings. Therefore, the qualifier values cannot be looked up.

44.1.1.2 Qualifier Order Support

A qualifier order is used to find the best match during lookup at runtime. The order of a qualifier varies from highest to lowest depending on the role of the qualifier in defining a more exact match. In Table 44–3, the state qualifier can have a higher order than the country qualifier, as a matching state indicates a more exact match.

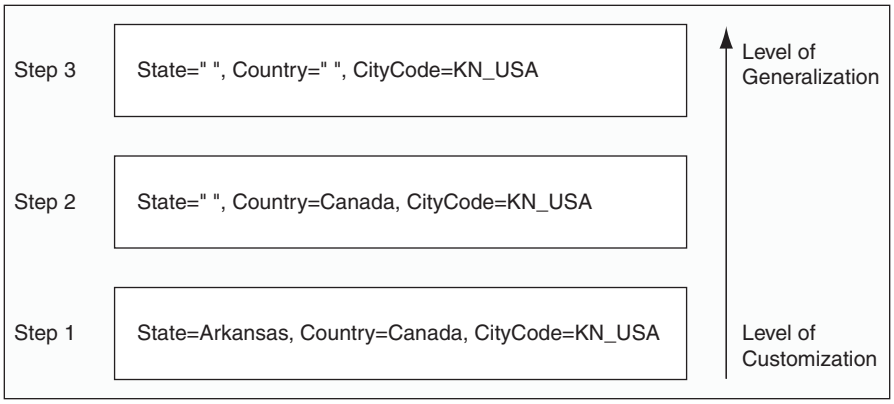
Domain value maps support hierarchical lookup. If you specify a qualifier value during a lookup and no exact match is found, then the lookup mechanism tries to find a more generalized match by setting the higher order qualifiers to a `" "`. It proceeds until a match is found, or until a match is not found with all qualifiers set to a `" "`. Figure 44–1 describes the hierarchical lookup performed for the following lookup in Table 44–3.

```
State=Arkansas, Country=Canada, CityCode=KN_USA
```

In this example, the `State` qualifier has a qualifier value of `1` and the `Country` qualifier has a qualifier value of `2`.

As shown in Figure 44–1, the lookup mechanism sets the higher order qualifier `State` to the exact lookup value `Arkansas` and uses `Canada|" "` for the lower order qualifier `Country`.

Figure 44–1 Hierarchical Lookup Example



When no match is found, the lookup mechanism sets the higher order qualifier `State` to a value of `" "` and sets the next higher qualifier `Country` to an exact value of `Canada`.

When no match is found, the lookup mechanism sets the value of the previous higher order qualifier `Country` to a value of `" "`. One matching row is found where `CityCode` is `KN_USA` and `Kensington` is returned as a value.

Table 44–4 provides a summary of these steps.

Table 44–4 Domain Value Map Lookup Result

State	Country	Short Value	Lookup Result
`Arkansas`	`CANADA|" "`	`KN_USA`	`No Match`
`" "`	`CANADA`	`KN_USA`	`No Match`
`" "`	`" "`	`KN_USA`	`Kensington`

44.1.1.3 One-to-Many Mapping Support

You can map one value to multiple values in a domain value map. For example, a domain value map for payment terms can contain a mapping of payment terms to three values, such as discount percentage, discount period, and net credit period, as shown in Table 44–5.

Table 44–5 One-to-Many Mapping Support

Payment Term	Discount Percentage	Discount Period	Net Credit Period
GoldCustomerPaymentTerm	10	20	30
SilverCustomerPaymentTerm	5	20	30
RegularPaymentTerm	2	20	30

44.2 Creating Domain Value Maps

You can create one or more domain value maps in a SOA composite application of Oracle JDeveloper, and then at runtime, use it to look up column values.

44.2.1 How to Create Domain Value Maps

You can create a domain value map by using the Create Domain Value Map(DVM) File dialog in Oracle JDeveloper.

To create a domain value map:

1. In the Application Navigator, right-click the project in which you want to create a domain value map and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **SOA Tier** node, and then select the **Transformations** category.
3. In the **Items** list, select **Domain Value Map(DVM)** and click **OK**.

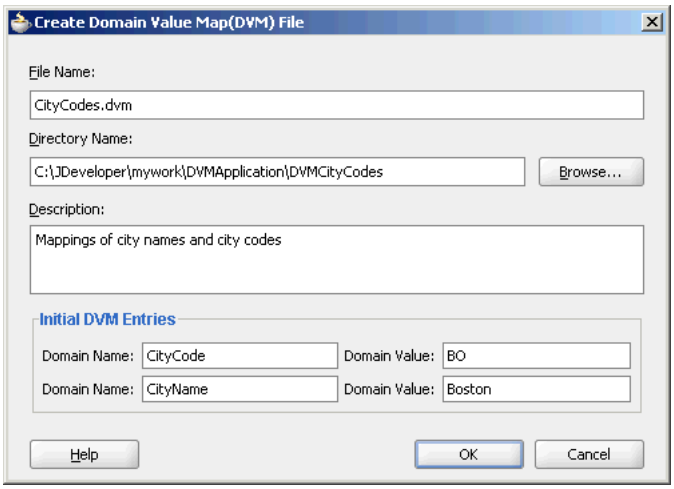
   The Create Domain Value Map(DVM) File dialog is displayed.

4. In the **File Name** field, enter the name of the domain value map file. For example, specify `CityCodes` to identify a domain value map for city names and city codes.
5. In the **Description** field, enter a description for the domain value map. For example, `Mappings of city names and city codes`. This field is optional.
6. In the **Domain Name** field, enter a name for each domain. For example, you can enter `CityCode` in one **Domain Name** field and `CityName` in another. Each domain name must be unique in a domain value map.

   Note: You can later add more domains to a domain value map by using the Domain Value Map Editor.

7. In the **Domain Value** field, enter a value corresponding to each domain. For example, enter `BO` for the `CityCode` domain and `Boston` for the `CityName` domain, as shown in Figure 44–2.

Figure 44–2 Populated Create Domain Value Map File Dialog



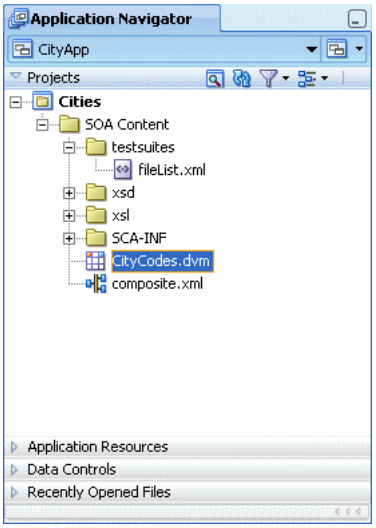
8. Click **OK**.

   The Domain Value Map Editor is displayed.

44.2.2 What Happens When You Create a Domain Value Map

A file with extension `.dvm` is created and appears in the Application Navigator, as shown in Figure 44–3.

Figure 44–3 A Domain Value Map File in Application Navigator



All `.dvm` files are based on the schema definition (XSD) file shown in Example 44–1.

Example 44–1 XSD File for Domain Value Map Files

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Copyright (c) 2006, Oracle. All rights reserved.  -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://xmlns.oracle.com/dvm"
                xmlns:tns="http://xmlns.oracle.com/dvm"
                elementFormDefault="qualified"
                attributeFormDefault="unqualified">
```

```
<xsd:element name="dvm">
    <xsd:annotation>
      <xsd:documentation>The Top Level Element
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="description" minOccurs="0" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>The DVM Description.  This is optional
            </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="columns">
          <xsd:annotation>
            <xsd:documentation>This element holds DVM's column List.
            </xsd:documentation>
          </xsd:annotation>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="column" minOccurs="2" maxOccurs="unbounded">
                <xsd:annotation>
                  <xsd:documentation>This represents a DVM Column
                  </xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                  <xsd:attribute name="name" use="required" type="xsd:string"/>
                  <xsd:attribute name="qualifier" default="false"
type="xsd:boolean"
 use="optional"/>
                  <xsd:attribute name="order" use="optional"
type="xsd:positiveInteger"/>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="rows" minOccurs="0">
          <xsd:annotation>
            <xsd:documentation>This represents all the DVM Rows.
            </xsd:documentation>
          </xsd:annotation>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="row" minOccurs="1" maxOccurs="unbounded">
                <xsd:annotation>
                  <xsd:documentation>
                    Each DVM row of values
                  </xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="cell" minOccurs="2" maxOccurs="unbounded"
                       type="xsd:string">
                      <xsd:annotation>
                        <xsd:documentation>This is the value for this row and for
 each column in the same order as defined in Columns.
                        </xsd:documentation>
                      </xsd:annotation>
```

```
                    </xsd:element>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="name" use="required" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:annotation>
    <xsd:documentation>This Schema is used to validate the DVM Document got for
 creation and
 update of a DVM.
    </xsd:documentation>
  </xsd:annotation>
</xsd:schema>
```

44.3 Editing a Domain Value Map

After you have created a domain value map, you can edit it and make adjustments to the presentation of data in the Domain Value Map Editor.

44.3.1 How to Add Columns to a Domain Value Map

A domain value map column defines the domain whose values you want to map with other domains.

To add a column to a domain value map:

1. Click **Add**.
2. Select **Add Column**.

   The Create DVM Column dialog is displayed.
3. In the **Name** field, enter a column name.
4. In the **Qualifier** field, select **True** to set this column as a qualifier. Otherwise, select **False**.
5. In the **Qualifier Order** field, enter a qualifier number. This field is enabled only if you selected **True** in the **Qualifier** field.
6. Click **OK**.

44.3.2 How to Add Rows to a Domain Value Map

A domain value map row contains the values of the domains.

To add a row to a domain value map:

1. In the Domain Value Map Editor, click **Add**.
2. Select **Add Row**.

44.4 Using Domain Value Map Functions

After creating a domain value map, you can use the XPath functions of the domain value map to look up appropriate values and populate the targets for the applications at runtime.

44.4.1 Understanding Domain Value Map Functions

You can use the `dvm:lookupValue` and `dvm:lookupValue1M` XPath functions to look up a domain value map for a single value or multiple values at runtime.

44.4.1.1 dvm:lookupValue

The `dvm:lookupValue` function returns a string by looking up the value for the target column in a domain value map, where the source column contains the given source value.

- Example 44–2 shows an example of `dvm:lookupValue` function syntax.

Example 44–2 dvm:lookupValue Function Syntax

```
dvm:lookupValue(dvmMetadataURI as string, SourceColumnName as string,
 SourceValue as string, TargetColumnName as string, DefaultValue as string) as
 string
```

Example 44–3 provides an example of `dvm:lookupValue` function use.

Example 44–3 dvm:lookupValue Function Use

```
dvm:lookupValue('cityMap.dvm','CityCodes','BO', 'CityNames',
'CouldNotBeFound')
```

- Example 44–4 shows another example of `dvm:lookupValue` function syntax.

Example 44–4 dvm:lookupValue Function Syntax

```
dvm:lookupValue(dvmMetadataURI as string, SourceColumnName as string,
SourceValue as string, TargetColumnName as string, DefaultValue as string,
(QualifierSourceColumn as string, QualifierSourceValue as string)*) as string
```

Example 44–5 provides another example of `dvm:lookupValue` function use.

Example 44–5 dvm:lookupValue Function Use

```
dvm:lookupValue ('cityMap.dvm','CityCodes','BO','CityNames',
 'CouldNotBeFound', 'State', 'Massachusetts')
```

Arguments

- `dvmMetadataURI` - The domain value map URI.
- `SourceColumnName` - The source column name.
- `SourceValue` - The source value (an XPath expression bound to the source document of the XSLT transformation).
- `TargetColumnName` - The target column name.
- `DefaultValue` - If the value is not found, then the default value is returned.
- `QualifierSourceColumn`: The name of the qualifier column.
- `QualifierSourceValue`: The value of the qualifier.

44.4.1.2 dvm:lookupValue1M

The `dvm:lookupValue1M` function returns an XML document fragment containing values for multiple target columns of a domain value map, where the value for the source column is equal to the source value. Example 44–6 provides details.

Example 44–6 dvm:lookupValue1M Function Syntax

```
dvm:lookupValue1M(dvmMetadataURI as string, SourceColumnName as string,
 SourceValue as string,(TargetColumnName as string)?)as nodeset
```

Arguments

- `dvmMetadataURI` - The domain value map URI.
- `SourceColumnName` - The source column name.
- `SourceValue` - The source value (an XPath expression bound to the source document of the XSLT transformation).
- `TargetColumnName` - The name of the target columns. At least one column name should be specified. The question mark symbol (`?`) indicates that you can specify multiple target column names.

Example 44–7 shows an example of `dvm:lookupValue1M` function use.

Example 44–7 dvm:lookupValue1M Function Use

```
dvm:lookupValue1M ('cityMap.dvm','CityCode','BO','CityName',
'CityNickName')
```

The result is shown in Example 44–8.

Example 44–8 dvm:lookupValue1M Function Result

```
<CityName>Boston</CityName>
<CityNickName>BeanTown</CityNickName>
```

44.4.2 How to Use Domain Value Map Functions in Transformations

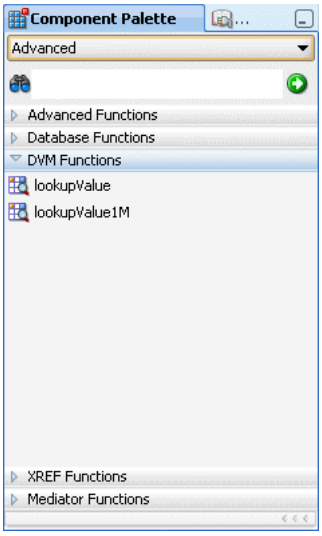
The domain value map functions can be used for transformations with a BPEL process service component or an Oracle Mediator service component. Transformations are performed by using the XSLT Mapper, which is displayed when you create an XSL file to transform the data from one XML schema to another.

For information about the XSLT Mapper, see Chapter 37, "Creating Transformations with the XSLT Mapper."

To use the lookupValue1M function in a transformation:

1. In the Application Navigator, double-click an XSL file to open the XSLT Mapper.
2. In the XSLT Mapper, expand the trees in the **Source** and **Target** panes.
3. In the Component Palette, click the down arrow, and then select **Advanced**.
4. Select **DVM Functions**, as shown in Figure 44–4.

Figure 44–4 Domain Value Map Functions in the Component Palette



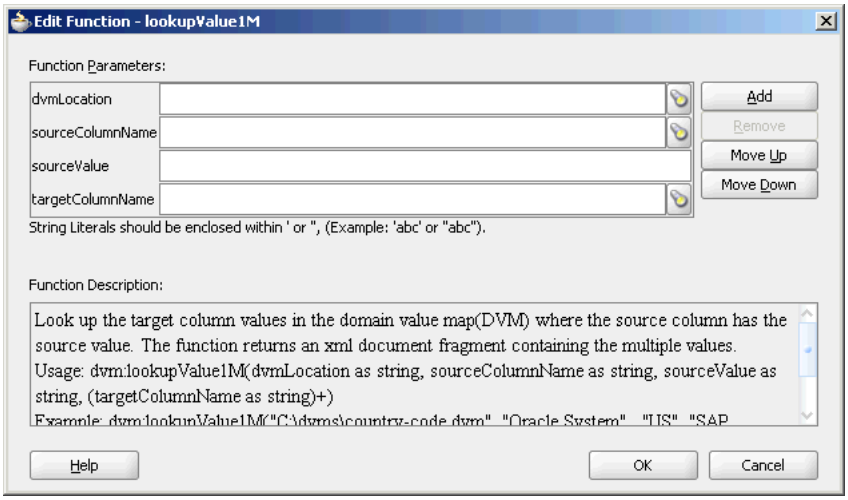
5. Drag and drop **lookupValue1M** onto the line that connects the source to the target.

   A **dvm:lookupValue1M** icon appears on the connecting line.

6. Double-click the **lookupValue1M** icon.

   The Edit Function – lookupValue1M dialog is displayed, as shown in Figure 44–5.

Figure 44–5 Edit Function – lookupValue1M Dialog



7. Specify values for the following fields in the Edit Function – lookupValue1M dialog:

   a. In the **dvmLocation** field, enter the location URI of the domain value map file or click **Browse** to the right of the **dvmLocation** field to select a domain value map file. You can select an already deployed domain value map from the metadata service (MDS) and also from the shared location in MDS. This can be done by selecting the Resource Palette.

   b. In the **sourceColumnName** field, enter the name of the domain value map column that is associated with the source element value, or click **Browse** to

select a column name from the columns defined for the domain value map you previously selected.

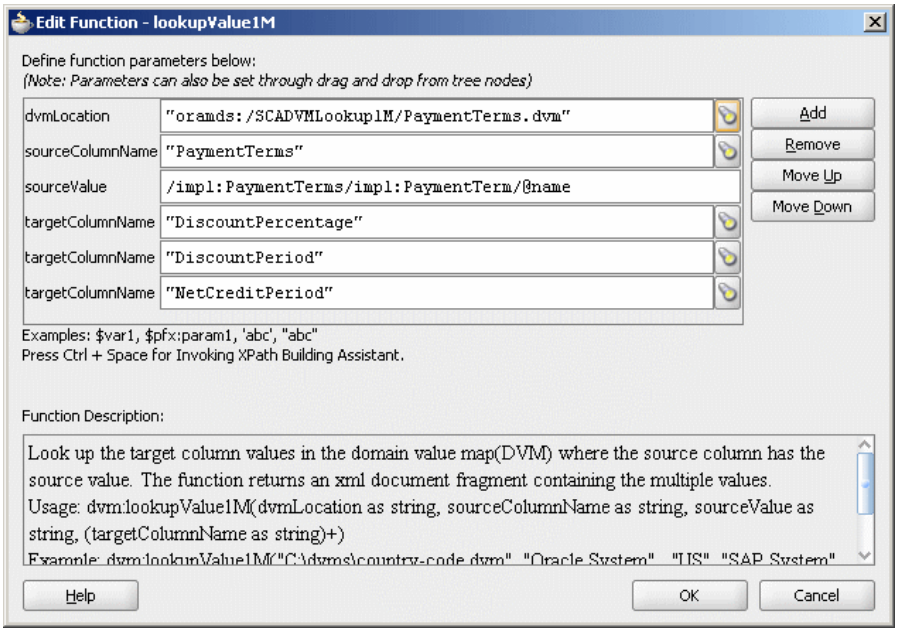
c. In the **sourceValue** field, enter a value or press **Ctrl-Space** to use the XPath Building Assistant. Press the up and down arrow keys to locate an object in the list, and press **Enter** to select an item.

d. In the **targetColumnName** field, enter the name of the domain value map column that is associated with the target element value, or click **Browse** to select the name from the columns defined for the domain value map you previously selected.

e. Click **Add** to add another column as the target column and then enter the name of the column.

A populated Edit Function - lookupValue1M dialog is shown in Figure 44–6.

Figure 44–6 Populated Edit Function – lookupValue1M Dialog



8. Click **OK**.

   The XSLT Mapper is displayed with the **lookupValue1M** function icon.

9. From the **File** menu, select **Save All**.

For more information about selecting deployed domain value maps, see Section 40.7.3, "Deploying and Using Shared Metadata Across SOA Composite Applications in Oracle JDeveloper."

44.4.3 How to Use Domain Value Map Functions in XPath Expressions

You can use the domain value map functions to create XPath expressions in the Expression Builder dialog. You can access the Expression Builder dialog through the Filter Expressions or the Assign Values functionality of an Oracle Mediator service component.

For information about the Assign Values functionality, see Section 19.2.2.9, "How to Assign Values."

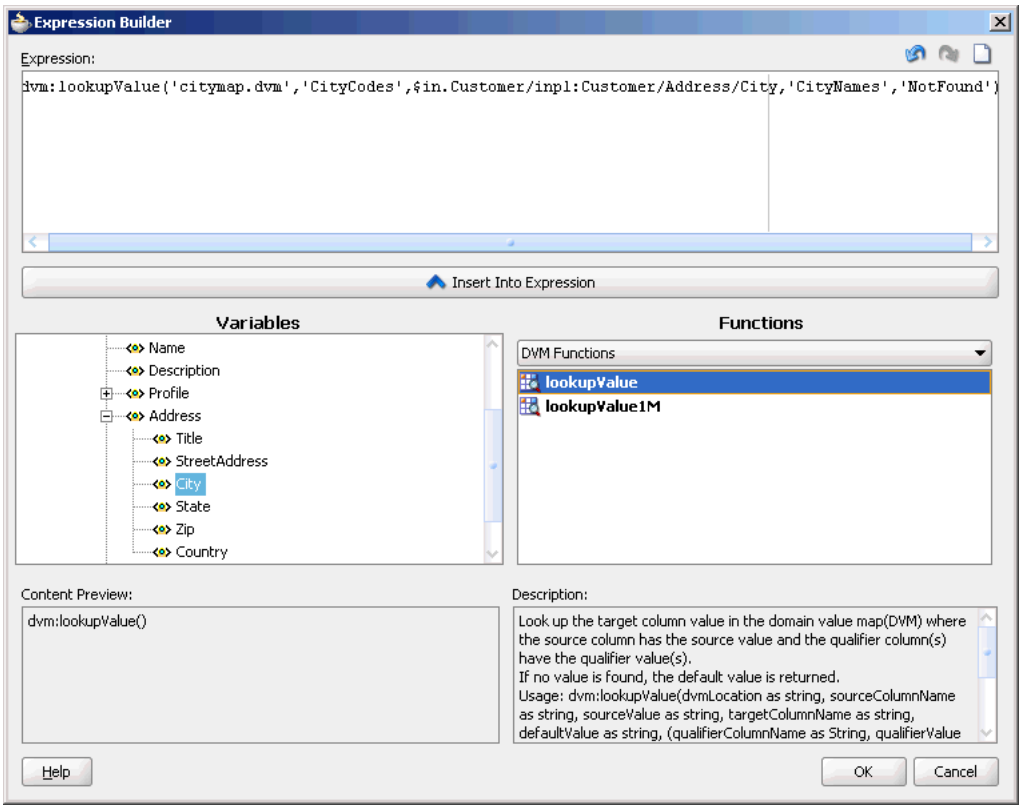
To use the lookupValue function in the Expression Builder dialog:

1. In the **Functions** list, select **DVM Functions**.
2. Double-click the **dvm:lookupValue** function to add it to the **expression** field.
3. Specify the various arguments of the **lookupValue** function. For example:

```
dvm:lookupValue('citymap.dvm','CityCodes',$in.Customer/inp1:Customer/Address/City,'CityNames','NotFound')
```

This expression, also shown in Figure 44–7, looks up a domain value map for the city name equivalent of a city code. The value of the city code depends on the value specified at runtime.

Figure 44–7 Domain Value Map Functions in the Expression Builder Dialog



44.4.4 What Happens at Runtime

At runtime, a BPEL process service component or an Oracle Mediator service component uses the domain value map to look up appropriate values.

44.5 Creating a Domain Value Map Use Case for a Hierarchical Lookup

This use case demonstrates the hierarchical lookup feature of domain value maps. The hierarchical lookup use case consists of the following steps:

1. Files are retrieved from a directory by an adapter service named ReadOrders.
2. The ReadOrders adapter service sends the file data to an Oracle Mediator named ProcessOrders.

3. The ProcessOrders Oracle Mediator then transforms the message to the structure required by the adapter reference. During transformation, Oracle Mediator looks up the UnitsOfMeasure domain value map for an equivalent value of the Common domain.
4. The ProcessOrders Oracle Mediator sends the message to an external reference named WriteOrders.
5. The WriteOrders reference writes the message to a specified output directory.

For downloading the sample files mentioned in this section, visit the following URL:

`https://soasamples.samplecode.oracle.com/#mediator`

44.5.1 How to Create the HierarchicalValue Use Case

This section provides the design-time tasks for creating, building, and deploying your SOA composite application. These tasks must be performed in the order in which they are presented.

44.5.1.1 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 `Hierarchical` and then click **Next**.

   The Name your project page appears.
5. In the **Project Name** field, enter `HierarchicalValue` 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 composite.
7. From the **File** menu, select **Save All**.

44.5.1.2 Task 2: How to Create a Domain Value Map

After creating an application and a project for the use case, you must create a domain value map.

To create a domain value map:

1. In the Application Navigator, right-click the **HierarchicalValue** project and select **New**.
2. In the New Gallery dialog, expand the **SOA Tier** node, and then select the **Transformations** category.
3. In the **Items** list, select **Domain Value Map(DVM)** and click **OK**.

The Create Domain Value Map(DVM) File dialog is displayed.

4. In the **File Name** field, enter `UnitsOfMeasure.dvm`.
5. In the **Domain Name** fields, enter `Siebel` and `Common`.
6. In the **Domain Value** field corresponding to the `Siebel` domain, enter `Ea`.
7. In the **Domain Value** field corresponding to the `Common` domain, enter `Each`.
8. Click **OK**.

   The Domain Value Map Editor is displayed.
9. Click **Add** and then select **Add Column**.

   The Create DVM Column dialog is displayed.
10. In the **Name** field, enter `TradingPartner`.
11. In the **Qualifier** list, select **true**.
12. In the **QualifierOrder** field, enter `1` and click **OK**.
13. Repeat Step 9 through Step 12 to create another qualifier named `StandardCode` with a qualifier order value of `2`.
14. Click **Add** and then select **Add Row**.

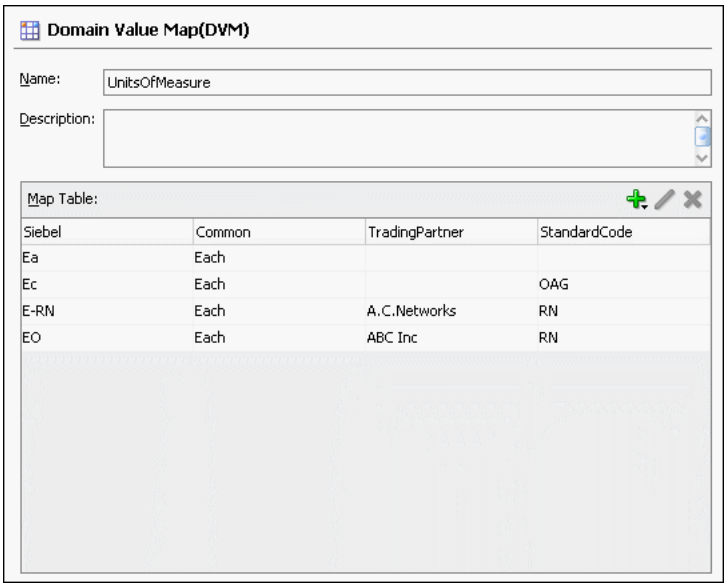
    Repeat this step to add two more rows.
15. Enter the information shown in Table 44–6 in the newly added rows of the domain value map table.

Table 44–6 Information for Rows of Domain Value Map Table

Siebel	Common	TradingPartner	StandardCode
EC	Each		OAG
E-RN	Each	A.C.Networks	RN
EO	Each	ABC Inc	RN

The Domain Value Map Editor appears, as shown in Figure 44–8.

Figure 44–8 UnitsOfMeasure Domain Value Map



16. From the **File** menu, select **Save All** and close the Domain Value Map Editor.

44.5.1.3 Task 3: How to Create a File Adapter Service

After creating the domain value map, you must create a file adapter service named ReadOrders to read the XML files from a directory.

> **Note:** Oracle Mediator may process the same file twice when run against Oracle 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.
3. If the Adapter Configuration wizard Welcome page appears, click **Next**.

   The Service Name page is displayed.
4. In the **Service Name** field, enter `ReadOrders` and then click **Next**.

   The Operation page is displayed.
5. In the **Operation Type** field, select **Read File** and then click **Next**.

   The File Directories page is displayed.
6. In the **Directory for Incoming Files (physical path)** field, enter the directory from which you want to read the files.
7. Click **Next**.

   The File Filtering page is displayed.
8. In the **Include Files with Name Pattern** field, enter `*.xml` and then click **Next.**

The File Polling page is displayed.

9. Change the **Polling Frequency** field value to **10 seconds** and then click **Next.**

   The Messages page is displayed.

10. Click **Search**.

    The Type Chooser dialog is displayed.

11. Click **Import Schema File**.

    The Import Schema File dialog is displayed.

12. Click **Search** and select the **Order.xsd** file in the `Samples` folder.

13. Click **OK**.

14. Expand the navigation tree to **Type Explorer > Imported Schemas > Order.xsd**.

15. Select **listOfOrder** and click **OK**.

16. Click **Next**.

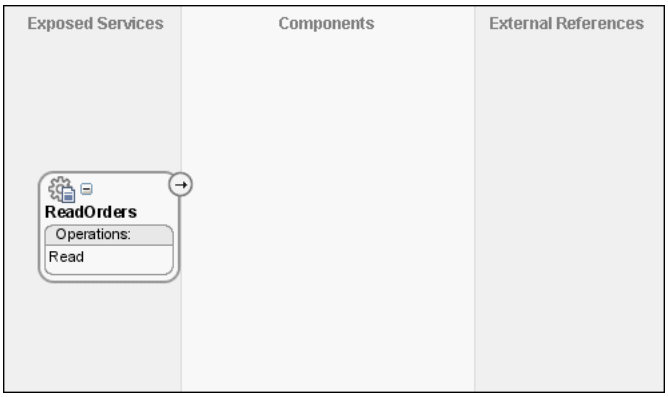
    The Finish page is displayed.

17. Click **Finish**.

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

    Figure 44–9 shows the **ReadOrders** service in the SOA Composite Editor.

Figure 44–9 ReadOrders Service in the SOA Composite Editor



44.5.1.4 Task 4: How to Create ProcessOrders Oracle Mediator Component

To create an Oracle Mediator named ProcessOrders:

1. Drag and drop a **Mediator** icon from the Component Palette to the **Components** section of the SOA Composite Editor.

   The Create Mediator dialog is displayed.

2. In the **Name** field, enter `ProcessOrders`.

3. From the **Template** list, select **Define Interface Later**.

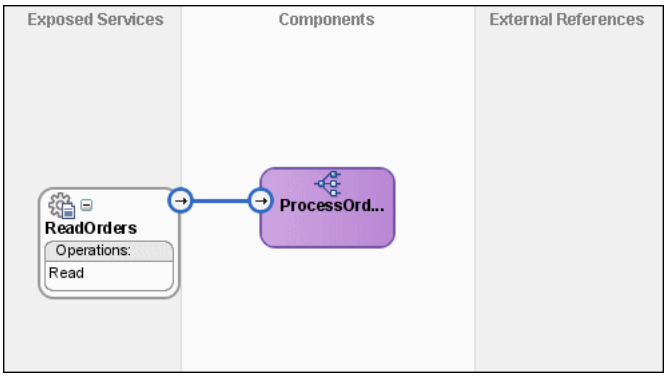
4. Click **OK**.

   An Oracle Mediator with name **ProcessOrders** is created.

5. In the SOA Composite Editor, connect the **ReadOrders** service to the **ProcessOrders** Oracle Mediator, as shown in Figure 44–10.

This specifies the file adapter service to invoke the **ProcessOrders** Oracle Mediator while reading a file from the input directory.

Figure 44–10 ReadOrders Service Connected to the ProcessOrders Oracle Mediator



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

44.5.1.5 Task 5: How to Create a 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 `WriteCommonOrder`.
5. Click **Next**.

   The Operation page is displayed.
6. In the **Operation Type** field, select **Write File**.
7. Click **Next**.

   The File Configuration page is displayed.
8. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory in which you want to write the files.
9. In the **File Naming Convention** field, enter `common_order_%SEQ%.xml` and click **Next**.

   The Messages page is displayed.
10. Click **Search**.

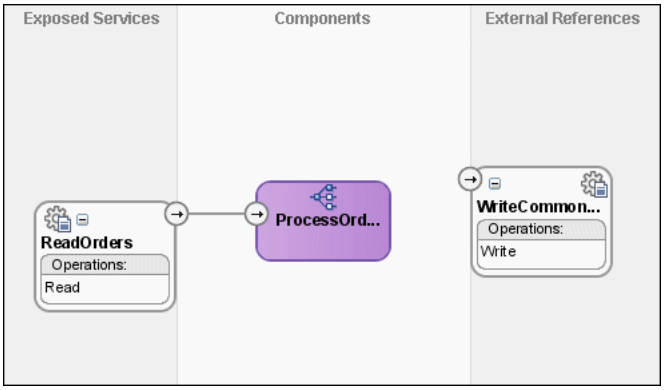
    The Type Chooser dialog is displayed.
11. Navigate to **Type Explorer > Project Schema Files > Order.xsd**, and then select **listOfOrder**.
12. Click **OK**.
13. Click **Next.**

The Finish page is displayed.

14. Click **Finish**.

    Figure 44–11 shows the **WriteCommonOrder** reference in the SOA Composite Editor.

Figure 44–11 WriteCommonOrder Reference in the SOA Composite Editor



15. From the **File** menu, select **Save All**.

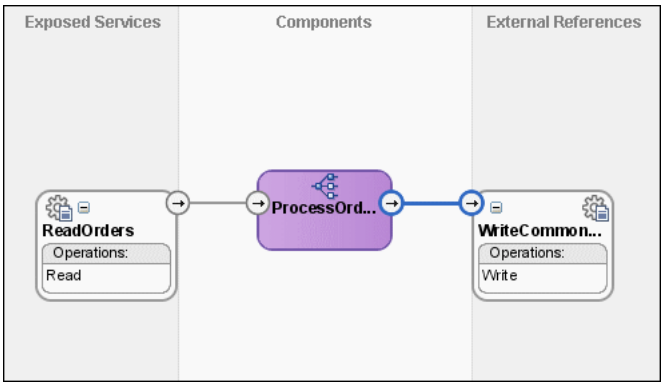
44.5.1.6 Task 6: How to Specify Routing Rules

You must specify the path that messages take from the **ReadOrders** adapter service to the external reference.

To specify routing rules:

1. Connect the **ProcessOrders** Oracle Mediator to the **WriteCommonOrder** reference, as shown in Figure 44–12.

Figure 44–12 ProcessOrders Oracle Mediator Connected to the WriteCommonOrder Reference



2. Double-click the **ProcessOrders** Oracle Mediator.
3. To the right of the **Transform Using** field, click the icon.

   The Request Transformation Map dialog is displayed.

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

   A **listOfOrder_To_listOfOrder.xsl** file is displayed in the XSLT Mapper.

5. Drag and drop the **imp1:listOfOrder** source element onto the **imp1:listOfOrder** target element.

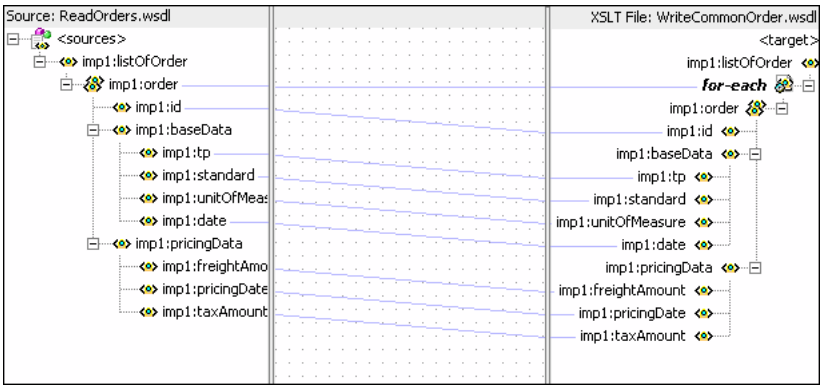
   The Auto Map Preferences dialog is displayed.

6. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names**.

7. Click **OK**.

   The **listOfOrder_To_listOfOrder.xsl** file appears, as shown in Figure 44–13.

Figure 44–13 imp1:listOfOrder To imp1:listOfOrder Transformation

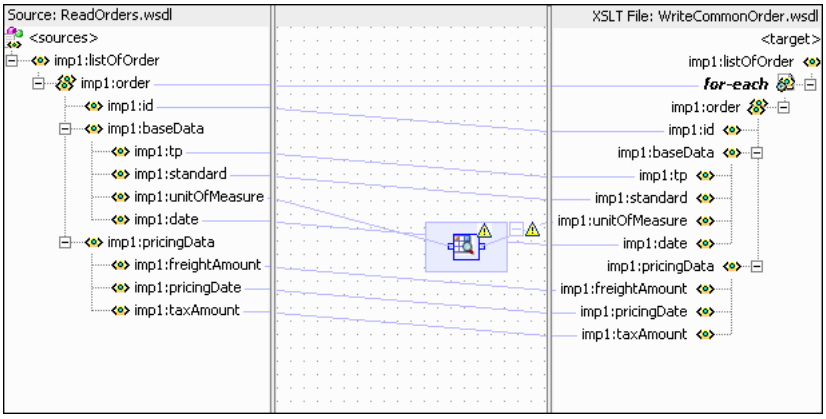


8. In the Component Palette, select **Advanced**.

9. Click **DVM Functions**.

10. Drag and drop **lookupValue** on the line connecting the **unitsOfMeasure** elements, as shown in Figure 44–14.

Figure 44–14 Adding lookupValue Function to imp1:listOfOrder To imp1:listOfOrder.xsl



11. Double-click the **lookupvalue** icon.

    The Edit Function-lookupValue dialog is displayed.

12. To the right of the **dvmLocation** field, click **Search**.

    The SOA Resource Lookup dialog is displayed.

13. Select **UnitsofMeasure.dvm** and click **OK**.

14. To the right of the **sourceColumnName** field, click **Search**.

The Select DVM Column dialog is displayed.

15. Select **Siebel** and click **OK**.
16. In the **sourceValue** column, enter the following:

    `/imp1:listOfOrder/imp1:order/imp1:baseData/imp1:unitOfMeasure`
17. To the right of the **targetColumnName** field, click **Search**.

    The Select DVM Column dialog is displayed.
18. Select **Common** and click **OK**.
19. In the **defaultValue** field, enter `"No_Value_Found"`.
20. Click **Add**.

    A **qualifierColumnName** row is added.
21. In the **qualifierColumnName** field, enter `"StandardCode"`.
22. Click **Add**.

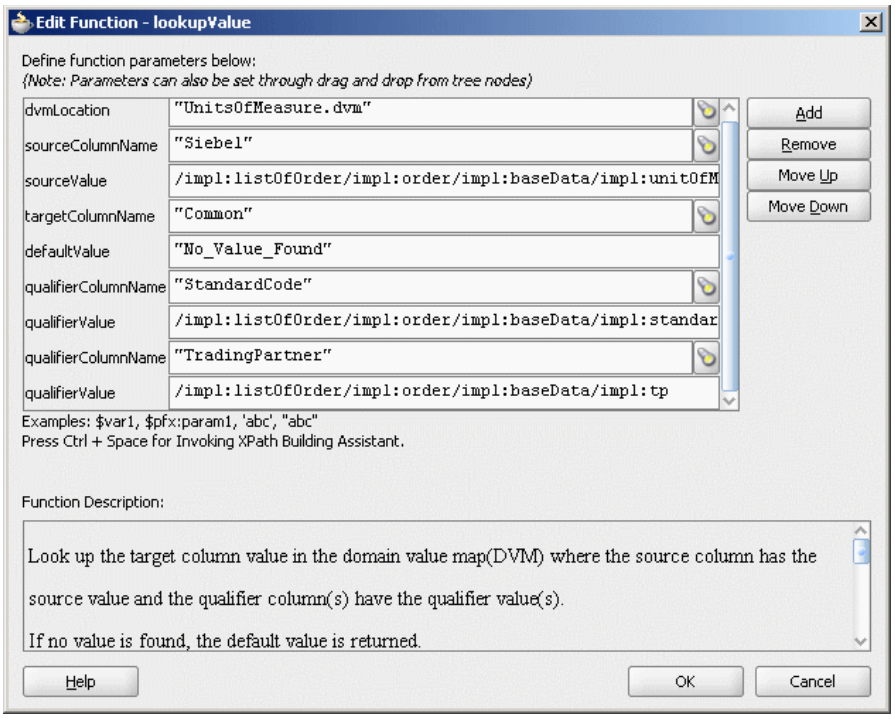
    A **qualifierValue** row is added.
23. In the **qualifierValue** field, enter the following:

    `/imp1:listOfOrder/imp1:order/imp1:baseData/imp1:standard`.
24. Click **Add** to insert another **qualifierColumnName** row.
25. In the **qualifierColumnName** field, enter `"TradingPartner"`.
26. Click **Add** to insert another **qualifierValue** row.
27. In the **qualifierValue** field, enter the following:

    `/imp1:listOfOrder/imp1:order/imp1:baseData/imp1:tp`.

    The Edit Function-lookupValue dialog appears, as shown in Figure 44–15.

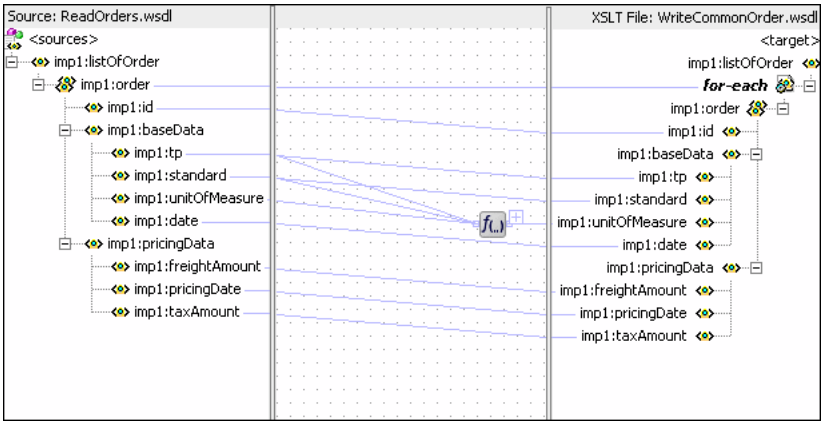
Figure 44–15 Edit Function-lookupValue Function Dialog: Hierarchical Lookup Use Case



28. Click **OK**.

The transformation appears, as shown in Figure 44–16.

Figure 44–16 Complete imp1:listOfOrder To imp1:listOfOrder Transformation



29. From the **File** menu, select **Save All** and close the **listOfOrder_To_listOfOrder.xsl** file at the top.

44.5.1.7 Task 7: 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."

44.5.1.8 Task 8: How to Deploy the Composite Application

Deploying the **HierarchicalValue** composite application to an application server consists of the following steps:

- Creating an application deployment profile.
- Deploying the application to the application server.

For detailed information about these steps, see Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper."

44.5.2 How to Run and Monitor the HierarchicalValue Application

After deploying the **HierarchicalValue** application, you can run it by copying the input XML file `sampleorder.xml` to the input folder. This file is available in the `samples` folder. On successful completion, a file named `common_order_1.xml` is written to the specified output directory.

For monitoring the running instance, you can use Oracle Enterprise Manager Fusion Middleware Control at the following URL:

`http://`*`hostname`*`:`*`port`*`/em`

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

For detailed information about these steps, see Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper."

44.6 Creating a Domain Value Map Use Case For Multiple Values

This use case demonstrates the integration scenario for using a domain value map lookup between two endpoints to look up multiple values. For example, if the inbound value is State, then the outbound values are Shortname of State, Language, and Capital. The multivalue lookup use case consists of the following steps:

1. Files are retrieved from a directory by an adapter service named readFile.
2. The readFile adapter service sends the file data to an Oracle Mediator named LookupMultiplevaluesMediator.
3. The LookupMultiplevaluesMediator Oracle Mediator then transforms the message to the structure required by the adapter reference. During transformation, Oracle Mediator looks up the multivalue domain value map for an equivalent value of the Longname and Shortname domains.
4. The LookupMultiplevaluesMediator Oracle Mediator sends the message to an external reference named writeFile.
5. The writeFile reference writes the message to a specified output directory.

For downloading the sample files mentioned in this section, visit the following URL:

`https://soasamples.samplecode.oracle.com/#mediator`

44.6.1 How to Create the Multivalue Use Case

This section provides the design-time tasks for creating, building, and deploying your SOA composite application. Perform these tasks in the order in which they are presented.

44.6.1.1 Task 1: How to Create an Oracle JDeveloper Application and Project

To create an Oracle JDeveloper application and 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 `Multivalue` and then click **Next**.

   The Name your project page appears.

5. In the **Project Name** field, enter `Multivalue` 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 project, and the SOA Composite Editor contains a blank composite.

7. From the **File** menu, select **Save All**.

44.6.1.2 Task 2: How to Create a Domain Value Map

After creating an application and a project for the use case, you must create a domain value map.

To create a domain value map:

1. In the Application Navigator, right-click the **Multivalue** project and select **New**.

2. In the New Gallery dialog, expand the **SOA Tier** node, and then select the **Transformations** category.

3. In the **Items** list, select **Domain Value Map(DVM)** and click **OK**.

   The Create Domain Value Map(DVM) File dialog is displayed.

4. In the **File Name** field, enter `multivalue.dvm`.

5. In the **Domain Name** fields, enter `Longname`, `Shortname`, `Language`, and `Capital`.

6. In the **Domain Value** field corresponding to the `Longname` domain, enter `Karnataka`.

7. In the **Domain Value** field corresponding to the `Shortname` domain, enter `KA`.

8. In the **Domain Value** field corresponding to the `Language` domain, enter `Kannada`.

9. In the **Domain Value** field corresponding to the `Capital` domain, enter `Bangalore`.

10. Click **OK**.

    The Domain Value Map Editor is displayed.

11. Click **Add** and then select **Add Row**.

Repeat this step to add two more rows.

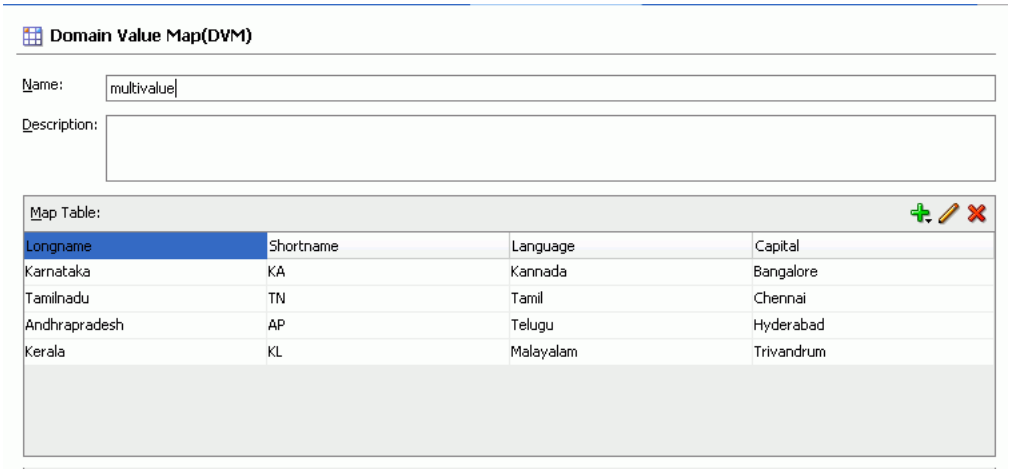
12. Enter the information shown in Table 44–7 in the newly added rows of the domain value map table:

Table 44–7 Information for Rows of Domain Value Map Table

Longname	Shortname	Language	Capital
Karnataka	KA	Kannada	Bangalore
Tamilnadu	TN	Tamil	Chennai
Andhrapradesh	AP	Telugu	Hyderbad
Kerala	KL	Malayalam	Trivandram

The Domain Value Map Editor appears, as shown in Figure 44–17.

Figure 44–17 Multivalue Domain Value Map



13. From the **File** menu, select **Save All** and close the Domain Value Map Editor.

44.6.1.3 Task 3: How to Create a File Adapter Service

After creating the domain value map, you must create a file adapter service named readFile to read the XML files from a directory.

> **Note:** Oracle Mediator may process the same file twice when run against Oracle RAC planned outages. This is because a file adapter is a non-XA compliant adapter. 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.
3. If the Adapter Configuration wizard Welcome page appears, click **Next**.

   The Service Name page is displayed.
4. In the **Service Name** field, enter `readFile` and then click **Next**.

The Adapter Interface page is displayed.

5. Click **Define from operation and schema (specified later)** and then click **Next**.

   The Operation page is displayed.

6. In the **Operation Type** field, select **Read File** and then click **Next**.

   The File Directories page is displayed.

7. In the **Directory for Incoming Files (physical path)** field, enter the directory from which you want to read the files.

8. Click **Next**.

   The File Filtering page is displayed.

9. In the **Include Files with Name Pattern** field, enter `*.xml` and then click **Next.**

   The File Polling page is displayed.

10. Change the **Polling Frequency** field value to **1 second** and then click **Next.**

    The Messages page is displayed.

11. Click **Search**.

    The Type Chooser dialog is displayed.

12. Click **Import Schema File**.

    The Import Schema File dialog is displayed.

13. Click **Search** and select the **input.xsd** file in the **Samples** folder.

14. Click **OK**.

15. Expand the navigation tree to **Type Explorer > Imported Schemas > input.xsd**.

16. Select **Root-Element** and click **OK**.

17. Click **Next**.

    The Finish page is displayed.

18. Click **Finish**.

19. From the **File** menu, select **Save All**.

    Figure 44–18 shows the **readFile** service in the SOA Composite Editor.

Figure 44–18 readFile Service in the SOA Composite Editor

44.6.1.4 Task 4: How to Create the LookupMultiplevaluesMediator Oracle Mediator

To create the LookupMultiplevaluesMediator Oracle Mediator:

1. Drag and drop a **Mediator** icon from the Component Palette to the **Components** section of the SOA Composite Editor.

   The Create Mediator dialog is displayed.

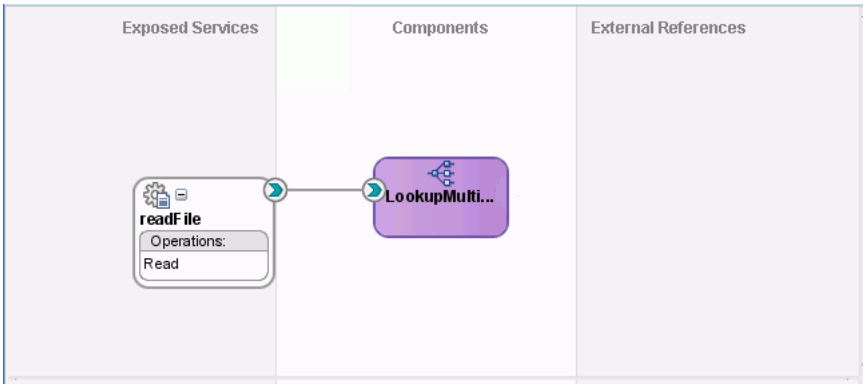
2. In the **Name** field, enter `LookupMultiplevaluesMediator`.
3. From the **Template** list, select **Define Interface Later**.
4. Click **OK**.

   An Oracle Mediator with the name **LookupMultiplevaluesMediator** is created.

5. In the SOA Composite Editor, connect the **readFile** service to the **LookupMultiplevaluesMediator** Oracle Mediator, as shown in Figure 44–19.

   This specifies the file adapter service to invoke the **LookupMultiplevaluesMediator** Oracle Mediator while reading a file from the input directory.

Figure 44–19 readFile Service Connected to the LookupMultiplevaluesMediator Oracle Mediator



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

44.6.1.5 Task 5: How to Create a 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 `writeFile` and then click **Next**.

   The Adapter Interface page is displayed.

5. Click **Define from operation and schema (specified later)** and then click **Next**.

   The Operation page is displayed.

6. Click **Next**.

   The Operation page is displayed.

7. In the **Operation Type** field, select **Write File**.

8. Click **Next**.

   The File Configuration page is displayed.

9. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory where you want to write the files.

10. In the **File Naming Convention** field, enter `multivalue_%SEQ%.xml` and click **Next**.

    The Messages page is displayed.

11. Click **Search**.

    The Type Chooser dialog is displayed.

12. Navigate to **Type Explorer > Project Schema Files > output.xsd**, and then select **Root-Element**.

13. Click **OK**.

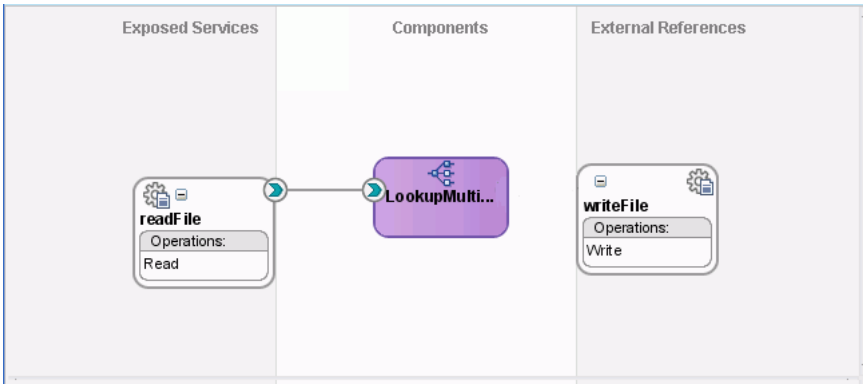
14. Click **Next.**

    The Finish page is displayed.

15. Click **Finish**.

    Figure 44–20 shows the **writeFile** reference in the SOA Composite Editor.

Figure 44–20 writeFile Reference in SOA Composite Editor



16. From the **File** menu, select **Save All**.

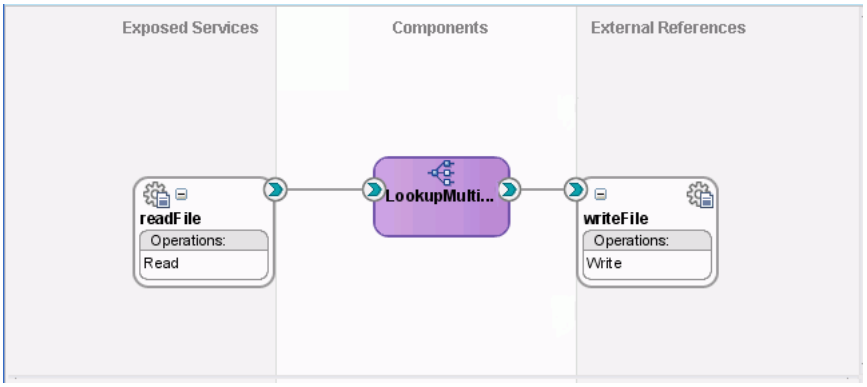
44.6.1.6 Task 6: How to Specify Routing Rules

You must specify the path that messages take from the readFile adapter service to the external reference.

To specify routing rules

1. Connect the **LookupMultiplevaluesMediator** Oracle Mediator to the **writeFile** reference, as shown in Figure 44–21.

Figure 44–21 LookupMultiplevaluesMediator Oracle Mediator Connected to the writeFile Reference



2. Double-click the **LookupMultiplevaluesMediator** Oracle Mediator.
3. To the right of the **Transform Using** field, click the icon.

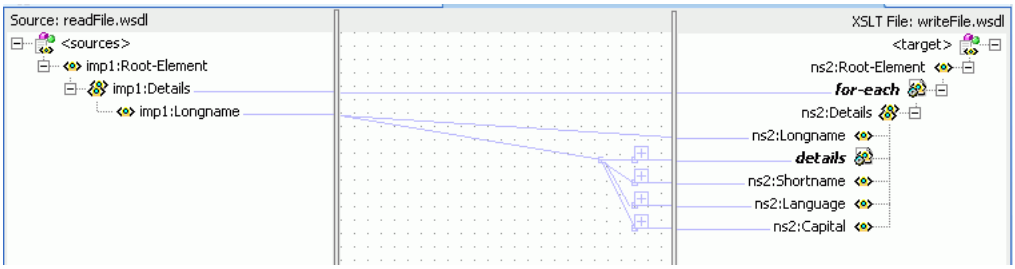
   The Request Transformation Map dialog is displayed.
4. Select **Create New Mapper File** and click **OK**.

   An **Input_To_Output_with_multiple_values_lookup.xsl** file is displayed in the XSLT Mapper.
5. Drag and drop the **imp1:Root-Element** source element to the **ns2:Root-Element** target element.

   The Auto Map Preferences dialog is displayed.
6. From the **During Auto Map** options list, deselect **Match Elements Considering their Ancestor Names**.
7. Click **OK**.

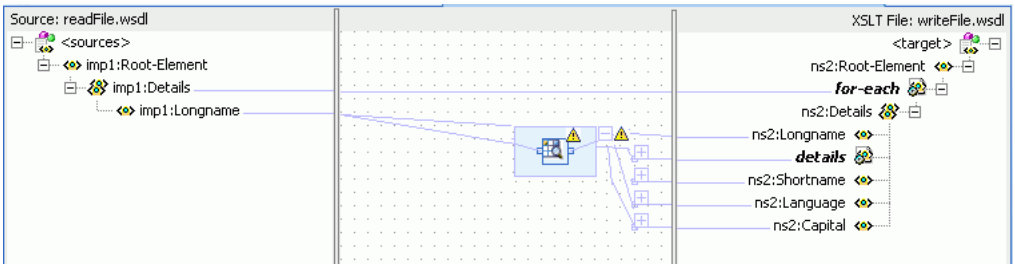
   The **Input_To_Output_with_multiple_values_lookup.xsl** file appears in the XSLT Mapper, as shown in Figure 44–22.

Figure 44–22 imp1:Root-Element To ns2:Root-Element Transformation



8. In the Component Palette, select **Advanced**.
9. Click **DVM Functions**.
10. Drag and drop **lookupValue1M** in the center panel, as shown in Figure 44–23.

Figure 44–23 Adding lookupValue Function to imp1:Root-Element to ns2:Root-Element



11. Double-click the **lookupvalue1M** icon.

    The Edit Function-lookupValue1M dialog is displayed.

12. To the right of the **dvmLocation** field, click **Search**.

    The SOA Resource Lookup dialog is displayed.

13. Select **multivalue.dvm** and click **OK**.

14. To the right of the **sourceColumnName** field, click **Search**.

    The Select DVM Column dialog is displayed.

15. Select **Longname** and click **OK**.

16. In the **sourceValue** column, enter the following:

    `/imp1:Root-Element/imp1:Details/imp1:Longname`.

17. To the right of the **targetColumnName** field, click **Search**.

    The Select DVM Column dialog is displayed.

18. Select **Shortname** and click **OK**.

19. Click **Add**.

    A **targetColumnName** row is added.

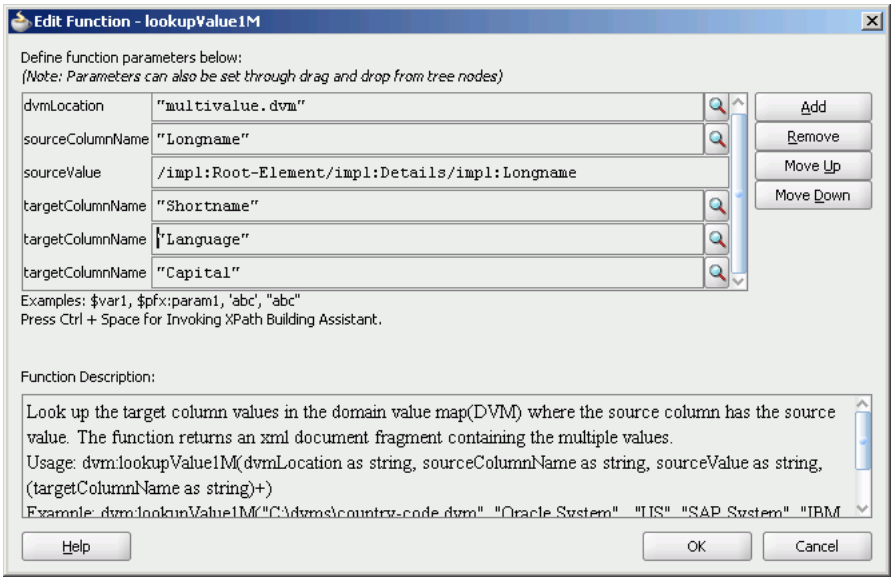
20. In the **targetColumnName** field, enter `"Language"`.

21. Click **Add** to insert another **targetColumnName** row.

22. In the **targetColumnName** field, enter `"Capital"`.

    The Edit Function-lookupValue dialog appears, as shown in Figure 44–24.

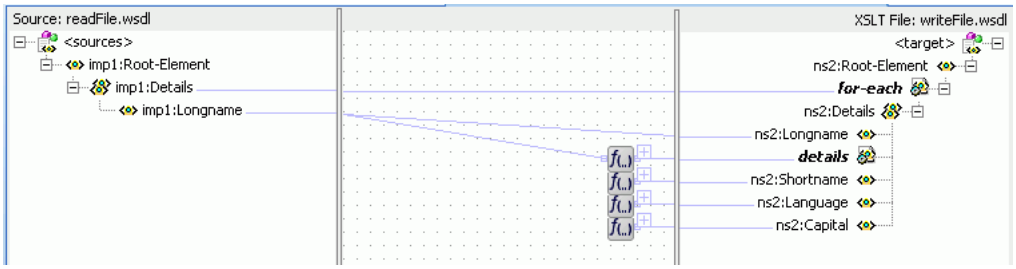
Figure 44–24 Edit Function-lookupValue Function Dialog: Multiple Value Lookup Use Case



23. Click **OK**.

The Transformation appears, as shown in Figure 44–25.

Figure 44–25 Complete imp1:Root-Element To ns2:Root-Element Transformation



24. From the **File** menu, select **Save All** and close the **Input_To_Output_with_multiple_values_lookup.xsl** file.

44.6.1.7 Task 7: 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."

44.6.1.8 Task 8: How to Deploy the Composite Application

Deploying the Multivalue composite application to an application server consists of the following steps:

- Creating an application deployment profile.
- Deploying the application to the application server.

For detailed information about these steps, see Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper."

44.6.2 How to Run and Monitor the Multivalue Application

After deploying the Multivalue application, you can run it by copying the input XML file `sampleinput.xml` to the input folder. This file is available in the `samples` folder. On successful completion, a file with name `multivalue_1.xml` is written to the specified output directory.

For monitoring the running instance, you can use Oracle Enterprise Manager Fusion Middleware Control at the following URL:

`http://`*`hostname`*`:`*`port`*`/em`

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

In Oracle Enterprise Manager Fusion Middleware Control, you can click **Multivalue** to see the project dashboard.

To view the detailed execution trail, click the instance ID in the instance column. The Flow Trace page is displayed.

45

Using Oracle SOA Composer with Domain Value Maps

Domain value maps enable you to map values from one vocabulary used in a given domain to another vocabulary used in a different domain. In earlier releases, for editing a domain value map at runtime, you first had to make the changes in Oracle JDeveloper, and then redeploy the domain value map in the application server. Oracle SOA Composer now offers support for editing domain value maps at runtime. Oracle SOA Composer is an EAR file, which is installed as part of Oracle SOA Suite installation. It enables you to manage domain value maps at runtime.

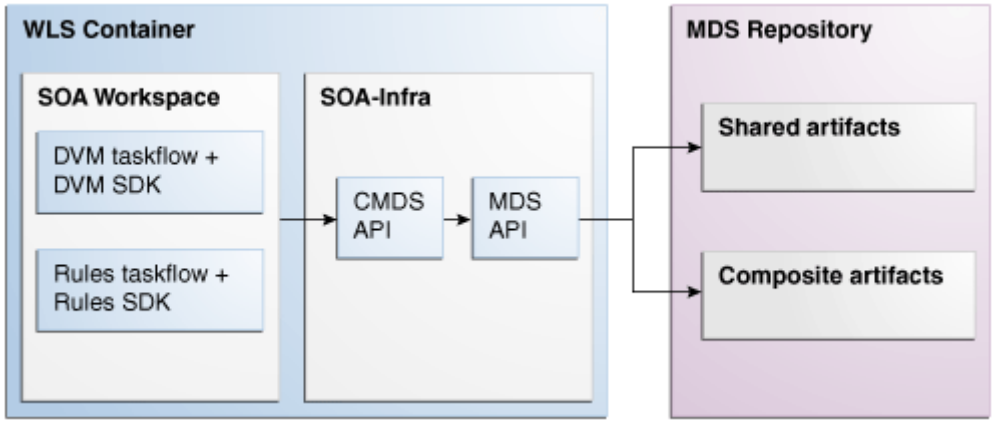
This chapter includes the following sections:

For more information about domain value maps, see Chapter 44, "Working with Domain Value Maps."

45.1 Introduction to Oracle SOA Composer

Oracle SOA Composer enables you to work with deployed domain value maps. Domain value map metadata can be associated either with a SOA composite application, or it can be shared across different composite applications. Figure 45–1 shows how Oracle SOA Composer enables you to access a domain value map from the Metadata Service (MDS) repository.

Figure 45–1 Oracle SOA Composer High-Level Deployment Topology



45.1.1 How to Log in to Oracle SOA Composer

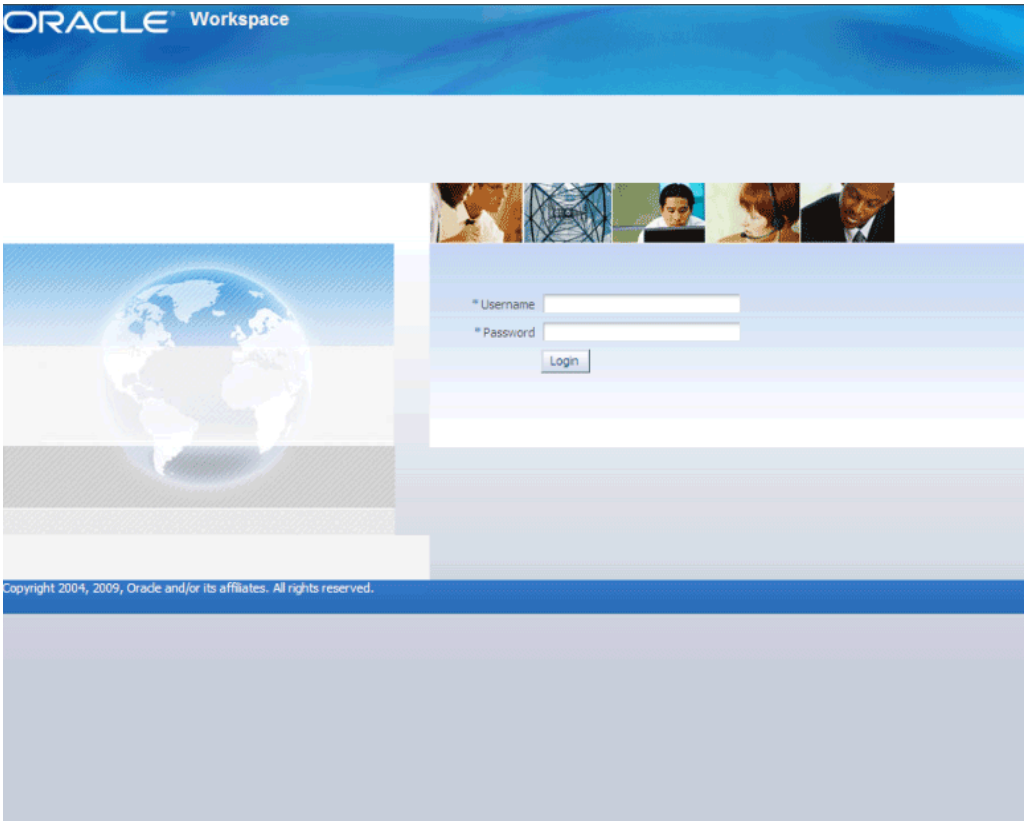
To log in to Oracle SOA Composer:

1. Access Oracle SOA Composer at the following location:

   `http://hostname:port/soa/composer`

   The Oracle SOA Composer Login page is displayed, as shown in Figure 45–2.

Figure 45–2 Oracle SOA Composer Login Page

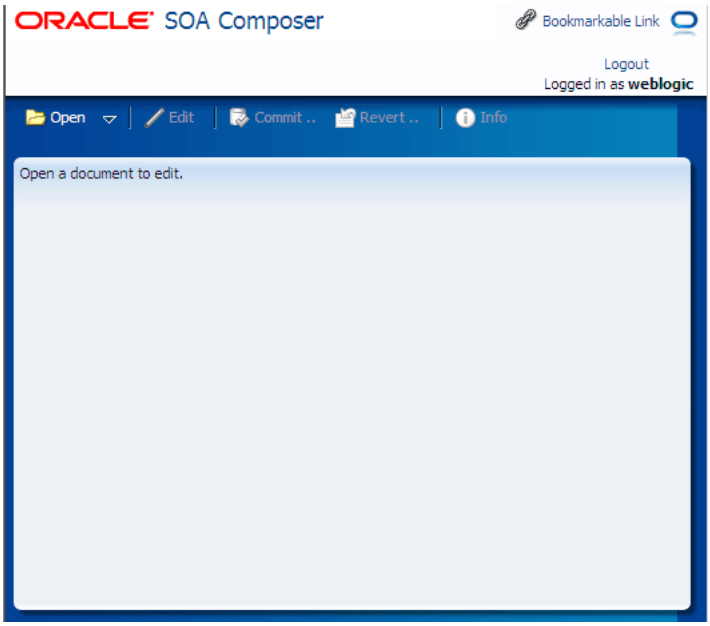


You must authenticate yourself by entering the login ID and password.

2. In the **Username** field, enter a user name.
3. In the **Password** field, enter a password.
4. Click **Login**.

After you log in to Oracle SOA Composer, you see the Oracle SOA Composer home page, as shown in Figure 45–3:

Figure 45–3 Oracle SOA Composer Home Page



You must have the `SOADesigner` application role to access Oracle SOA Composer metadata. By default, all the users with Oracle Enterprise Manager Fusion Middleware Control administrator privileges have this role. If you log in to Oracle SOA Composer without this role, you see the following message:

```
Currently logged in user is not authorized to modify SOA metadata.
```

For information about adding the `SOADesigner` application role to users without administrator privileges, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

45.2 Viewing Domain Value Maps at Runtime

You can view domain value maps at runtime. Perform the following steps to open and view a domain value map.

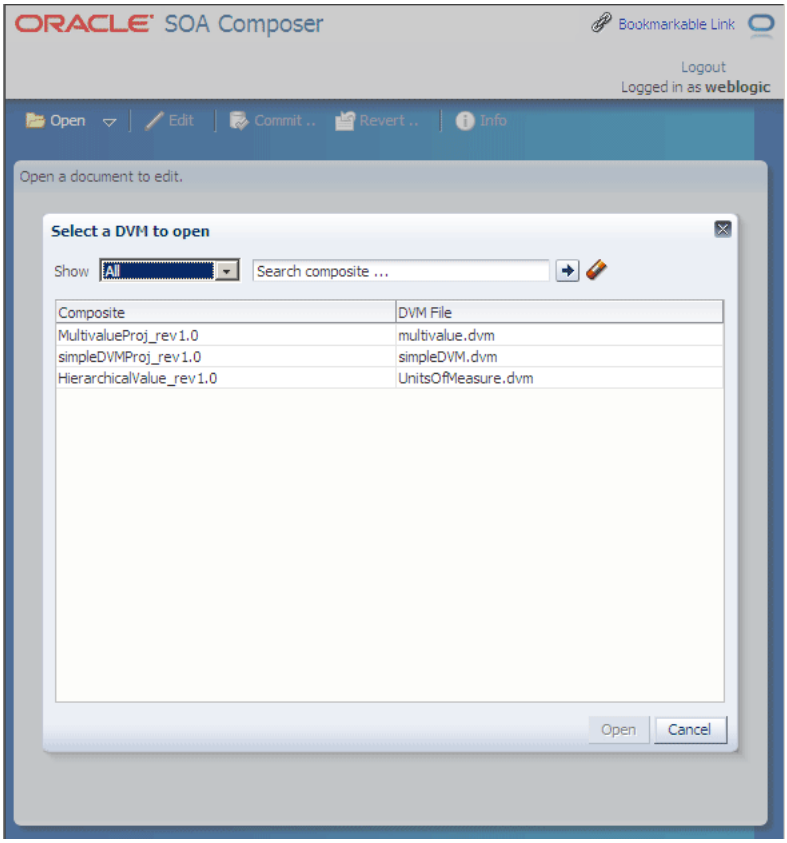
45.2.1 How To View Domain Value Maps at Runtime

To view domain value maps at runtime:

1. From the **Open** menu, select **Open DVM**.

   The Select a DVM to open dialog appears, as shown in Figure 45–4:

Figure 45–4 Select a DVM to Open Dialog



You can also select a document from the **My Edits** option that displays recently opened documents.

Note: Alternatively, you can also search for a domain value map by entering the name of the composite application containing the domain value map file in the **Search composite** field and then clicking the **Search** icon to the right of the field.

---

2. Select a domain value map and click **Open**. You can also double-click a domain value map to open it.

   The selected domain value map opens in view mode.

You can click **Bookmarkable Link** to get a direct link to the selected domain value map. The **Info** button provides more information on the selected domain value map.

45.3 Editing Domain Value Maps at Runtime

You can edit domain value maps at runtime. By default, domain value maps open in view mode. To edit a domain value map, you must change the mode to an edit session by clicking the **Edit** menu item.

45.3.1 How to Edit Domain Value Maps at Runtime

The domain value map opens in an edit session.

45.3.1.1 Adding Rows

To add rows:

You can add rows by performing the following steps:

1. Click **Add Domain Values**.

   The Add Domain Values dialog is displayed.

2. Enter values and click **OK**.

   The entered values are added to the domain value map.

45.3.1.2 Editing Rows

To edit rows:

You can edit rows by performing the following steps:

1. Select the row to edit.
2. Click **Edit Domain Values**.

   The Edit Domain Values dialog is displayed.

3. Edit the values as required and click **OK**.

45.3.1.3 Deleting Rows

To delete rows:

You can delete rows by performing the following steps:

1. Select the rows to delete.
2. Click **Delete Domain Values**.

45.4 Saving Domain Value Maps at Runtime

Every time a domain value map is opened in an edit session, a sandbox is created per domain value map, per user. If you save your changes, then the changes are saved in your sandbox.

45.4.1 How to Save Domain Value Maps at Runtime

To save domain value maps at runtime:

1. Click the **Save** menu item to save your changes. If your changes are saved successfully, you receive a notification message.

   You can also revert a domain value map to the last saved state.

2. Click the **Revert** menu item. A confirmation dialog is displayed.
3. Click **Yes** to revert your changes.

45.5 Committing Changes at Runtime

You must commit the changes for saving them permanently. Once you commit the changes, runtime picks up the changes and saves them in the MDS repository. In a session, you can also save your changes without committing them. In such a case, the

domain value map remains in the saved state. You can reopen the domain value map and commit the changes later.

45.5.1 How to Commit Changes at Runtime

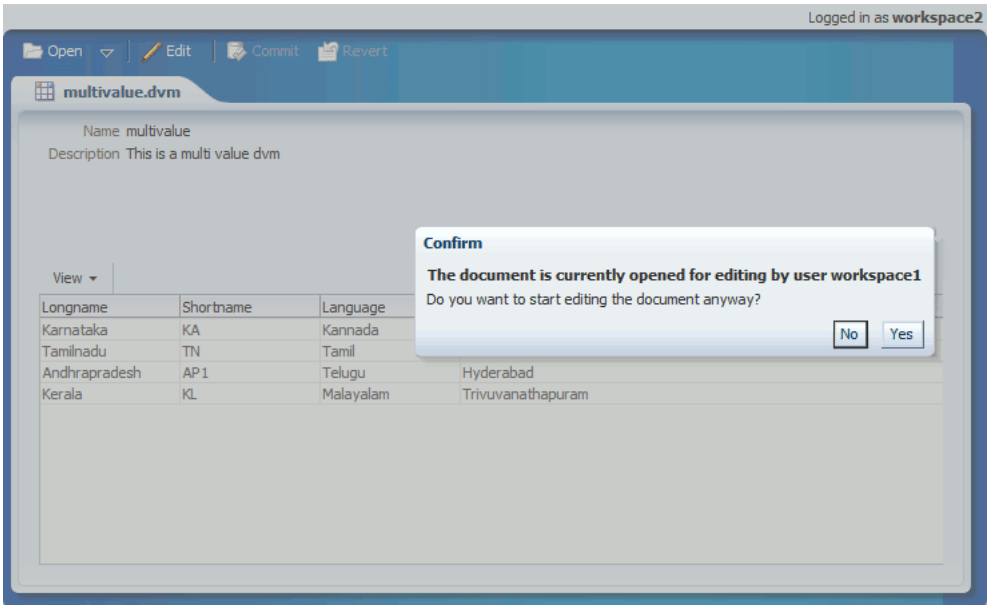
To commit changes at runtime:

1. Click the **Commit** menu option. A confirmation dialog is displayed.
2. Click **Yes** to commit your changes.

45.6 Detecting Conflicts

Oracle SOA Composer detects conflicts that can occur among concurrent users. If you open a domain value map that is being edited by another user, then you see a warning, as shown in Figure 45–5.

Figure 45–5 Confirm Dialog for Concurrent Users of a Domain Value Map



However, if you still want to edit the domain value map, you can click **Yes** and make the modifications.

If the other user makes changes to the domain value map and commits the changes, you receive a notification message while trying to commit your changes.

If you click **Yes** and commit your changes, then the changes made by the other user are overwritten by your changes.

46

Working with Cross References

This chapter describes how to use the cross referencing feature of Oracle SOA Suite to associate identifiers for equivalent entities created in different applications.

This chapter includes the following sections:

46.1 Introduction to Cross References

Cross references enable you to dynamically map values for equivalent entities created in different applications.

> **Note:** The cross referencing feature enables you to dynamically integrate values between applications, whereas domain value maps enable you to specify values at design time and edit values at runtime. For more information about domain value maps, see Chapter 44, "Working with Domain Value Maps" and Chapter 45, "Using Oracle SOA Composer with Domain Value Maps."

When you create or update objects in one application, you may also want to propagate the changes to other applications. For example, when a new customer is created in an SAP application, you may want to create a new entry for the same customer in your Oracle E-Business Suite application named EBS. However, the applications that you are integrating may be using different entities to represent the same information. For example, for each new customer in an SAP application, a new row is inserted in its `Customer` database with a unique identifier such as `SAP_001`. When the same information is propagated to an Oracle E-Business Suite application and a Siebel application, the new row should be inserted with different identifiers such as `EBS_1001` and `SBL001`. In such cases, you need some type of functionality to map these identifiers with each other so that they can be interpreted by different applications to be referring to the same entity. This can be done by using cross references.

46.2 Introduction to Cross Reference Tables

Cross references are stored in the form of tables. Table 46–1 shows a cross reference table containing information about customer identifiers in different applications.

Table 46–1 Cross Reference Table Sample

SAP	EBS	SBL
SAP_001	EBS_1001	SBL001
SAP_002	EBS_1002	SBL002

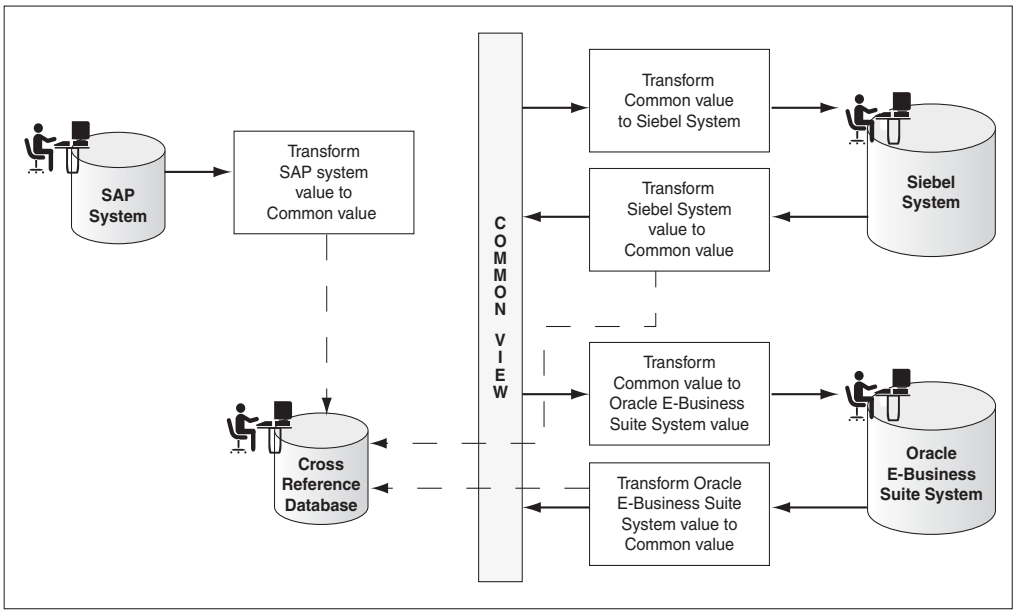
The identifier mapping is also required when information about a customer is updated in one application and the changes must be propagated to other applications. You can integrate different identifiers by using a common value integration pattern, which maps to all identifiers in a cross reference table. For example, you can add one more column named `Common` to the cross reference table shown in Table 46–1. The updated cross reference table then appears, as shown in Table 46–2.

Table 46–2 Cross Reference Table with Common Column

SAP	EBS	SBL	Common
SAP_001	EBS_1001	SBL001	CM001
SAP_002	EBS_1002	SBL002	CM002

Figure 46–1 shows how you can use common value integration patterns to map identifiers in different applications.

Figure 46–1 Common Value Integration Pattern Example



A cross reference table consists of two parts: metadata and actual data. The metadata is saved as the `.xref` file created in Oracle JDeveloper, and is stored in the Metadata Services (MDS) repository as an XML file. By default, the actual data is stored in the `XREF_DATA` table of the database in the SOA Infrastructure database schema. You can

also generate a custom database table for each cross reference entity. The database table depends on the metadata of the cross reference entity.

Consider the following two cross reference entities:

- ORDER with cross reference columns SIEBEL, COMMON, and EBS, as shown in Table 46–3
- CUSTOMER with cross reference columns EBS, COMMON, and PORTAL, as shown in Table 46–4

Table 46–3 ORDER Table

Column Name	SIEBEL	COMMON	EBS
Column Value	SBL_101	COM_100	EBS_002
Column Value		COM_110	EBS_012

Table 46–4 CUSTOMER Table

Column Name	EBS	COMMON	PORTAL
Column Value	EBS_201	COM_200	P2002

If you chose to save all the runtime data in one generic table, then the data is stored in the XREF_DATA table, as shown in Table 46–5.

Table 46–5 XREF_DATA Table

XREF_TABLE_NAME	XREF_COLUMN_NAME	ROW_NUMBER	VALUE	IS_DELETED
ORDER	SIEBEL	100012345	SBL_101	N
ORDER	COMMON	100012345	COM_100	N
ORDER	EBS	100012345	EBS_002	N
ORDER	COMMON	110012345	COM_110	N
ORDER	EBS	110012345	EBS_012	N
CUSTOMER	EBS	200212345	EBS_201	N
CUSTOMER	COMMON	200212345	COM_200	N
CUSTOMER	PORTAL	200212345	P2002	N

This approach has the following advantages:

- The process of adding, removing, and modifying the columns of the cross reference entities is simple.
- The process of creating and deleting cross reference entities from an application is straightforward.

However, this approach has the following disadvantages:

- A large number of rows are generated in the database because each cross reference cell is mapped to a different row in the database. This reduces the performance of the queries.
- In the generic table, the data for the columns XREF_TABLE_NAME and XREF_COLUMN_NAME is repeated across a large number of rows.

To overcome these problems, you can generate a custom database table for each cross reference entity. The custom database tables depend on the metadata of the cross reference entities. For example, for the `XREF_ORDER` table and `XREF_CUSTOMER` table, you can generate the custom database tables shown in Table 46–6 and Table 46–7.

Table 46–6 XREF_ORDER Table

ROW_ID	SIEBEL	COMMON	EBS
100012345	SBL_101	COM_100	EBS_002
110012345		COM_110	EBS_012

Table 46–7 XREF_CUSTOMER Table

ROW_ID	EBS	COMMON	PORTAL
200212345	EBS_201	COM_200	P2002

This approach requires you to execute Data Definition Language (DDL) scripts to generate the custom database tables. For more information about custom database tables, see Section 46.3.3, "How to Create Custom Database Tables."

46.3 Creating and Modifying Cross Reference Tables

You can create cross references tables in a SOA composite application and then use it with a BPEL process service component or an Oracle Mediator service component during transformations.

46.3.1 How to Create Cross Reference Metadata

To create cross reference metadata:

1. In Oracle JDeveloper, select the SOA project in which you want to create the cross reference.
2. Right-click the project and select **New**.

   The New Gallery dialog is displayed.
3. Select **SOA Tier** from the **Categories** section, and then select **Transformations**.
4. Select **Cross Reference(XREF)** from the **Items** section.
5. Click **OK**.

   The Create Cross Reference(XREF) File dialog is displayed.
6. In the **File Name** field, specify the name of the cross reference file. For example, specify `Customer`.

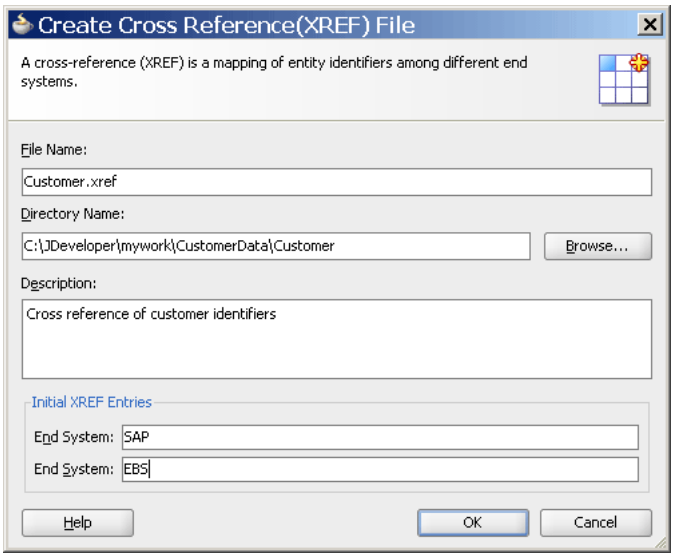
   A cross reference name is used to uniquely identify a cross reference table. Two cross reference tables cannot have same name in the cross reference repository. The cross reference file name is the name of the cross reference table with an extension of `.xref`.
7. In the **Description** field, enter a description for the cross reference. For example:

   ```
   Cross reference of Customer identifiers.
   ```
8. In the **End System** fields, enter the end system names.

The end systems map to the cross reference columns in a cross reference table. For example, you can change the first end system name to `SAP` and the second end system name to `EBS`. Each end system name must be unique within a cross reference.

A sample Create Cross Reference(XREF) File dialog is displayed in Figure 46–2.

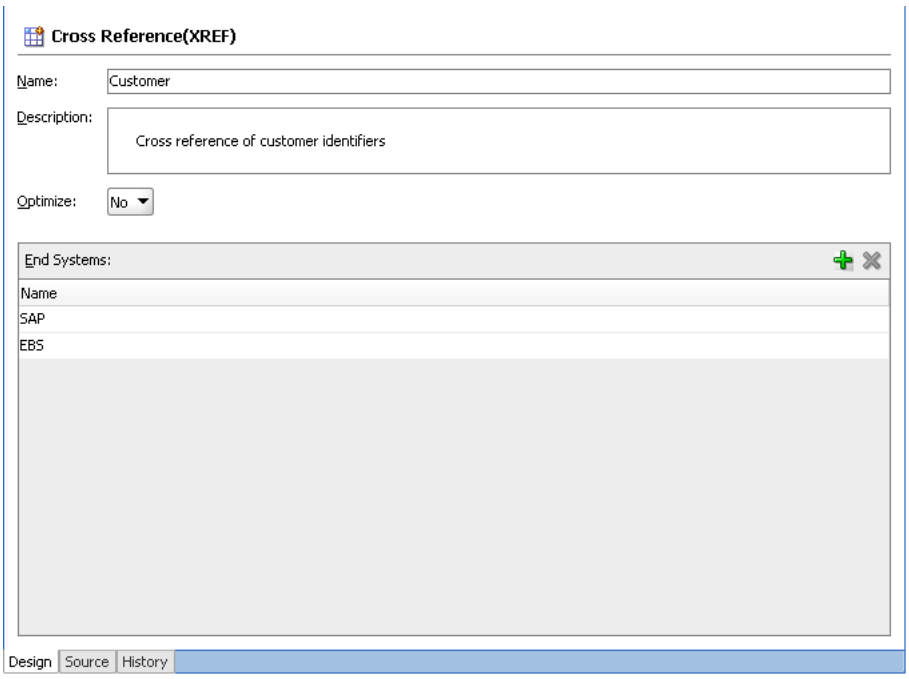
Figure 46–2 Create Cross Reference(XREF) File Dialog



9. Click **OK**.

   The Cross Reference Editor is displayed, as shown in Figure 46–3. You can use this editor to modify the cross reference.

Figure 46–3 Cross Reference Editor



46.3.2 What Happens When You Create a Cross Reference

A file with extension `.xref` gets created and appears in the Application Navigator. All `.xref` files are based on the schema definition (XSD) file shown in Example 46–1.

Example 46–1 Cross Reference XSD File

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xmlns.oracle.com/xref"
        xmlns:tns="http://xmlns.oracle.com/xref" elementFormDefault="qualified">
  <element name="xref" type="tns:xrefType"/>
  <complexType name="xrefType">
    <sequence>
      <element name="table">
        <complexType>
          <sequence>
            <element name="description" type="string" minOccurs="0"
                     maxOccurs="1"/>
            <element name="columns" type="tns:columnsType" minOccurs="0"
                     maxOccurs="1"/>
            <element name="rows" type="tns:rowsType" maxOccurs="1"
                     minOccurs="0"/>
          </sequence>
          <attribute name="name" type="string" use="required"/>
        </complexType>
      </element>
    </sequence>
  </complexType>

  <complexType name="columnsType">
    <sequence>
      <element name="column" minOccurs="1" maxOccurs="unbounded">
        <complexType>
          <attribute name="name" type="string" use="required"/>
        </complexType>
      </element>
    </sequence>
  </complexType>

  <complexType name="rowsType">
    <sequence>
      <element name="row" minOccurs="1" maxOccurs="unbounded">
        <complexType>
          <sequence>
            <element name="cell" minOccurs="1" maxOccurs="unbounded">
              <complexType>
                <attribute name="colName" type="string" use="required"/>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</schema>
```

46.3.3 How to Create Custom Database Tables

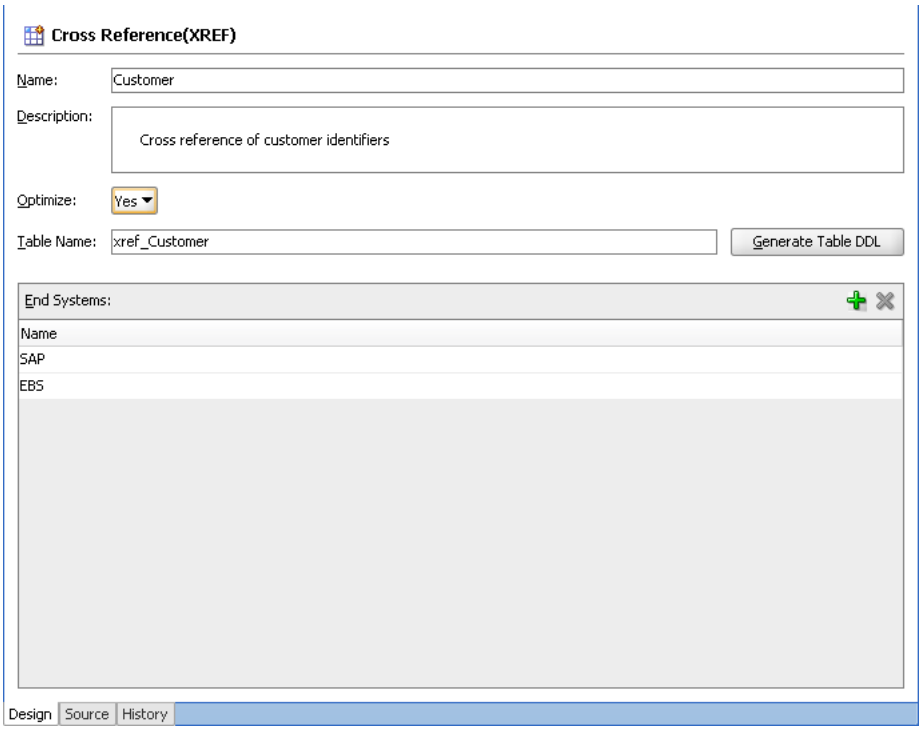
As mentioned previously, all the runtime data by default gets stored in the XREF_DATA table. If you want to create custom database tables, then perform the following steps.

To create custom database tables:

1. From the **Optimize** list, select **Yes** in the Cross Reference Editor.

   The name of the custom database table to be generated is displayed in the **Table Name** field, as shown in Figure 46–4.

Figure 46–4 Generating Custom Database Tables



   The **Table Name** field is editable and you can change the name of the custom table. The custom database table name should be prefixed with xref_. If you do not prefix your table name with xref_, then while generating the table, you receive the following error message:

```
Table name should begin with 'xref_' and cannot be 'xref_data' or
'xref_deleted_data' which are reserved table names for XREF runtime.
```

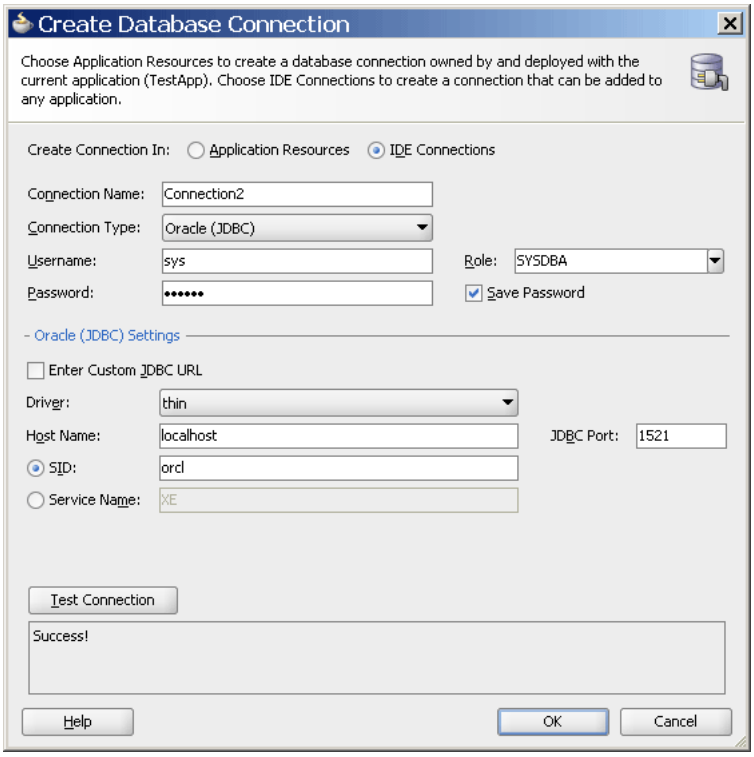
2. Click **Generate Table DDL**. The Optimize XREF dialog is displayed.
3. Select the **Generate Drop DDL** checkbox to drop the table and associated indexes, if a table with the same name already exists. If you select this option and click **Run**, then the Running Drop DDL Warning dialog is displayed with the following message:

```
Running the Drop DDL will remove the table and indexes, do you want to
continue?
```

4. Click **Run**. The Run Table DDL dialog is displayed.
5. From the **Connection** list, select the database connection to use.

If there is no available connection, then click **Create a new database connection** to open the Create Database Connection dialog, as shown in Figure 46–5. If you want to edit an existing connection, then select the connection and click **Edit selected database connection** to open the Edit Database Connection dialog.

Figure 46–5 Create Database Connection Dialog



6. Enter all the required details and click **OK**. The **Connection** list of the Run Table DDL dialog is now populated.
7. Click **OK** on the Run Table DDL dialog to run the DDL script.

   The Table DDL Run Results dialog displays the execution status of your DDL scripts.

For custom database tables, two additional attributes, namely `mode` and `dbtable`, are added to the schema definition mentioned in Section 46.3.2, "What Happens When You Create a Cross Reference." They are added for the `table` element in the following way:

```
<attribute name="mode" type="string" default="generic" />
<attribute name="dbtable" type="string" default="xref_data"/>
```

46.3.4 How to Add an End System to a Cross Reference Table

To add an end system to a cross reference table:

1. Click **Add**.

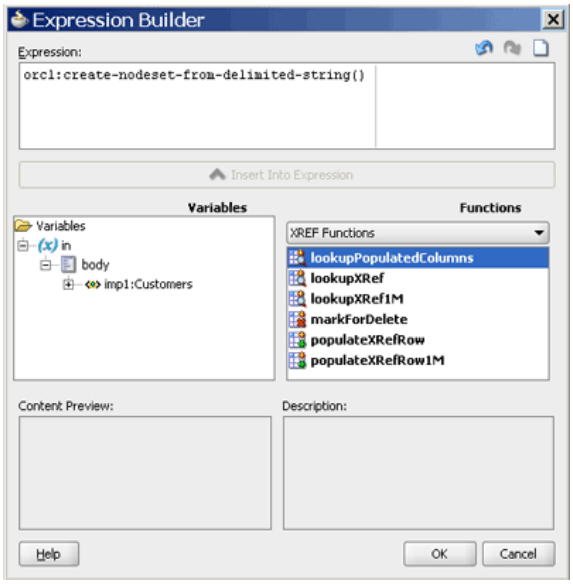
   A new row is added.
2. Double-click the newly-added row.
3. Enter the end system name. For example, `SBL`.

46.4 Populating Cross Reference Tables

You can create a cross reference table in a SOA composite application in Oracle JDeveloper and then use it to look up column values at runtime. However, before using a cross reference to look up a particular value, you must populate it at runtime. You can use the cross reference XPath functions to populate the cross-reference tables. The XPath functions enable you to populate a cross reference column, perform lookups, and delete a column value. These XPath functions can be used in the Expression Builder dialog to create an expression or in the XSLT Mapper to create transformations. For example, you can use the `xref:populateXRefRow` function to populate a cross reference column with a single value and the `xref:populateXRefRow1M` function to populate a cross reference column with multiple values.

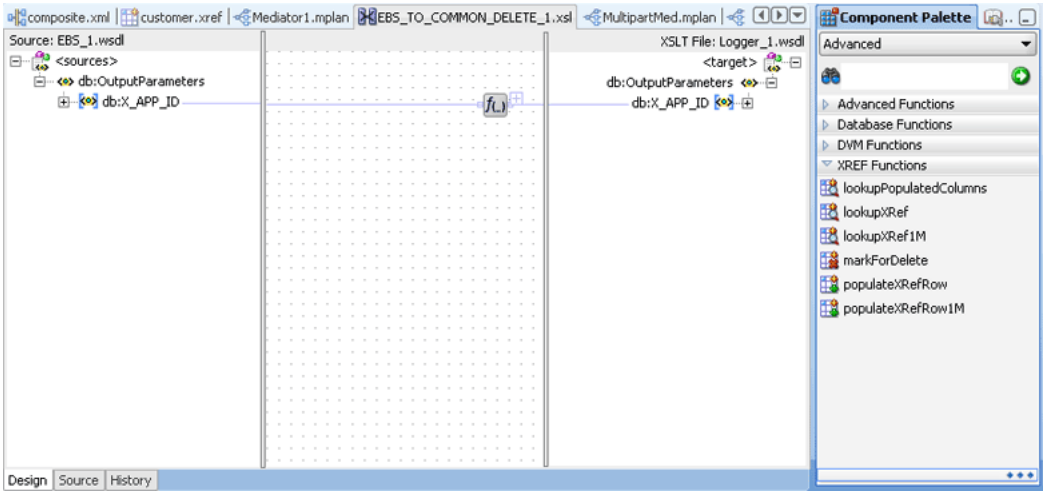
You can access the Expression Builder dialog through an assign activity, an XSL transformation, or the filtering functionality of a BPEL process service component or an Oracle Mediator service component. Figure 46–6 shows how you can select the cross reference functions in the Expression Builder dialog.

Figure 46–6 Expression Builder Dialog with Cross Reference Functions



The XSLT Mapper is displayed when you create an XSL file to transform data from one XML schema to another. Figure 46–7 shows how you can select the cross reference functions in the XSLT Mapper.

Figure 46–7 XSLT Mapper Dialog with Cross Reference Functions



A cross reference table must be populated at runtime before using it. By default, the data is stored in the `XREF_DATA` table under the SOA Infrastructure database schema. You can use the `xref:populateXRefRow` function to populate a cross reference column with a single value and the `xref:populateXRefRow1M` function to populate a cross reference column with multiple values.

Note: You can also store the data in a different database schema by configuring a data source in the following way:

- The JNDI name of the data source should be `jdbc/xref`.
- The *`ORACLE_HOME`*`/rcu/integration/soainfra/sql/xref/createschema_xref_oracle.sql` file should be loaded to create the `XREF_DATA` table in this data source.

46.4.1 About the xref:populateXRefRow Function

The `xref:populateXRefRow` function populates a cross reference column with a single value. The `xref:populateXRefRow` function returns a string value, which is the cross reference value being populated. For example, as shown in Table 46–8, the `Order` table has the following columns: `EBS`, `Common`, and `SBL` with values `E100`, `100`, and `SBL_001` respectively.

Table 46–8 Cross Reference Table with Single Column Values

EBS	Common	SBL
E100	100	SBL_001

The syntax of the `xref:populateXRefRow` function is shown in Example 46–2.

Example 46–2 xref:populateXRefRow Function

```
xref:populateXRefRow(xrefLocation as string, xrefReferenceColumnName as string,
 xrefReferenceValue as string, xrefColumnName as string, xrefValue as string, mode
 as string) as string
```

Parameters

- `xrefLocation`: The cross reference table URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be populated.
- `xrefValue`: The value to be populated in the column.
- `mode`: The mode in which the `xref:populateXRefRow` function populates the column. You can specify any of the following values: `ADD`, `LINK`, or `UPDATE`. Table 46–9 describes these modes.

Table 46–9 xref:populateXRefRow Function Modes

Mode	Description	Exception Reasons
`ADD`	Adds the reference value and the value to be added. For example, the following mode: `xref:populateXRefRow("customers.xref", "EBS","EBS100", "Common","CM001", "ADD")` Adds the reference value `EBS100` in the `ESB` reference column and the value `CM001` in the `Common` column.	Exceptions can occur for the following reasons: ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ The value being added is not unique across that column for that table. ■ The column for that row already contains a value. ■ The reference value exists.
`LINK`	Adds the cross reference value corresponding to the existing reference value. For example, the following mode: `xref:populateXRefRow("customers.xref", "Common","CM001","SBL","SBL_ 001","LINK")` Links the value `CM001` in the `Common` column to the `SBL_001` value in the `SBL` column.	Exceptions can occur for the following reasons: ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ The reference value is not found. ■ The value being linked exists in that column for that table.
`UPDATE`	Updates the cross reference value corresponding to an existing reference column-value pair. For example, the following mode: `xref:populateXRefRow("customers.xref", "SBL","SBL_001", "SBL","SBL_ 1001","UPDATE")` Updates the value `SBL_001` in the `SBL` column to the value `SBL_1001`.	Exceptions can occur for the following reasons: ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ Multiple values are found for the column being updated. ■ The reference value is not found. ■ The column for that row does not have a value.

Note: The mode parameter values are case-sensitive and should be specified in upper case only, as shown in Table 46–9.

Table 46–10 describes the `xref:populateXRefRow` function modes and exception conditions for these modes.

Table 46–10 xref:populateXRefRow Function Results with Different Modes

Mode	Reference Value	Value to be Added	Result
ADD	Absent	Absent	Success
	Present	Absent	Exception
	Present	Present	Exception
LINK	Absent	Absent	Exception
	Present	Absent	Success
	Present	Present	Exception
UPDATE	Absent	Absent	Exception
	Present	Absent	Exception
	Present	Present	Success

46.4.2 About the xref:populateXRefRow1M Function

Two values in an end system can correspond to a single value in another system. In such a scenario, you should use the `xref:populateXRefRow1M` function to populate a cross reference column with a value. For example, as shown in Table 46–11, the `SAP_001` and `SAP_0011` values refer to one value of the `EBS` and `SBL` applications. To populate columns such as `SAP`, you can use the `xref:populateXRefRow1M` function.

Table 46–11 Cross Reference Table with Multiple Column Values

SAP	EBS	SBL
SAP_001	EBS_1001	SBL001
SAP_0011		
SAP_002	EBS_1002	SBL002

The syntax of the `xref:populateXRefRow1M` function is shown in Example 46–3.

Example 46–3 xref:populateXRefRow1M Function

```
xref:populateXRefRow1M(xrefLocation as string, xrefReferenceColumnName as string,
 xrefReferenceValue as string, xrefColumnName as string, xrefValue as string, mode
 as string) as string
```

Parameters

- `xrefLocation`: The cross reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be populated.
- `xrefValue`: The value to be populated in the column.

- `mode`: The mode in which the `xref:populateXRefRow` function populates the column. You can specify either of the following values: `ADD` or `LINK`. Table 46–12 describes these modes:

Table 46–12 xref:populateXRefRow1M Function Modes

Mode	Description	Exception Reasons
`ADD`	Adds the reference value and the value to be added. For example, the following mode: `xref:populateXRefRow1M("customers.xref","EBS","EBS_1002", "SAP","SAP_0011","ADD")` Adds the reference value `EBS_1002` in the reference column `EBS` and the value `SAP_0011` in the `SAP` column.	Exceptions can occur for the following reasons: ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ The value being added is not unique across that column for that table. ■ The reference value exists.
`LINK`	Adds the cross reference value corresponding to the existing reference value. For example, the following mode: `xref:populateXRefRow1M("customers.xref","EBS","EBS_1002", "SAP","SAP_002","LINK")` Links the value `SAP_002` in the `SAP` column to the `EBS_1002` value in the `EBS` column.	Exceptions can occur for the following reasons: ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ The reference value is not found. ■ The value being added is not unique across the column for that table.

Table 46–13 describes the `xref:populateXRefRow1M` function modes and exception conditions for these modes.

Table 46–13 xref:populateXRefRow1M Function Results with Different Modes

Mode	Reference Value	Value to be Added	Result
`ADD`	`Absent`	`Absent`	Success
	`Present`	`Absent`	Exception
	`Present`	`Present`	Exception
`LINK`	`Absent`	`Absent`	Exception
	`Present`	`Absent`	Success
	`Present`	`Present`	Exception

46.4.3 How to Populate a Column of a Cross Reference Table

To populate a column of a cross reference table:

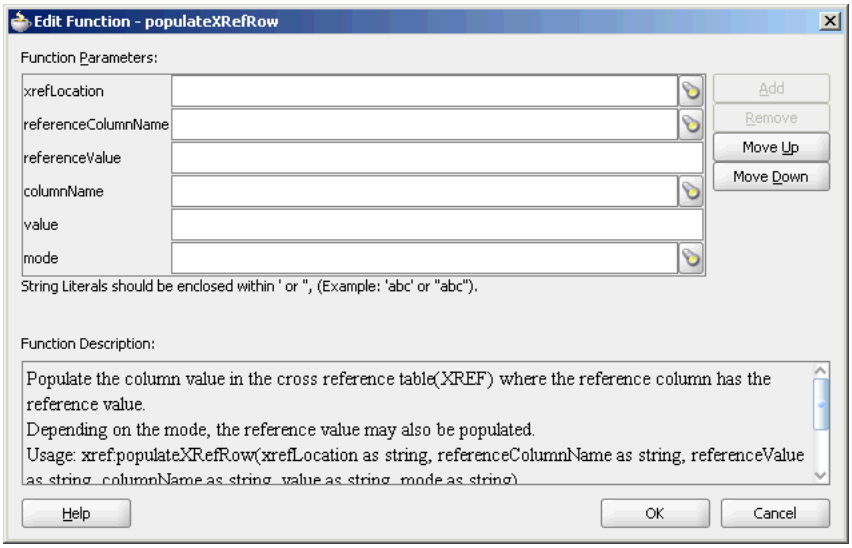
1. In the XSLT Mapper, expand the trees in the **Source** and **Target** panes.

2. Drag and drop a source element to a target element.
3. In the Component Palette, select **Advanced**.
4. Select **XREF Functions**.
5. Drag and drop the **populateXRefRow** function to the line that connects the source object to the target object.

   A **populateXRefRow** icon appears on the connecting line.
6. Double-click the **populateXRefRow** icon.

   The Edit Function – populateXRefRow dialog is displayed, as shown in Figure 46–8.

Figure 46–8 Edit Function – populateXRefRow Dialog



7. Specify the following values for the fields in the Edit Function – populateXRefRow dialog:

   a. In the **xrefLocation** field, enter the location URI of the cross reference file.

      Click **Browse** to the right of the **xrefLocation** field to select the cross reference file. You can select an already-deployed cross reference from MDS and also from a shared location in MDS using the Resource Palette.

   b. In the **referenceColumnName** field, enter the name of the cross reference column.

      Click **Browse** to the right of the **referenceColumnName** field to select a column name from the columns defined for the cross reference you previously selected.

   c. In the **referenceValue** field, you can manually enter a value or press **Ctrl-Space** to launch the XPath Building Assistant. Press the up and down keys to locate an object in the list and press **Enter** to select that object.

   d. In the **columnName** field, enter the name of the cross reference column.

      Click the **Browse** icon to the right of the **columnName** field to select a column name from the columns defined for the cross reference you previously selected.

e. In the **value** field, you can manually enter a value or press **Ctrl-Space** to launch the XPath Building Assistant.

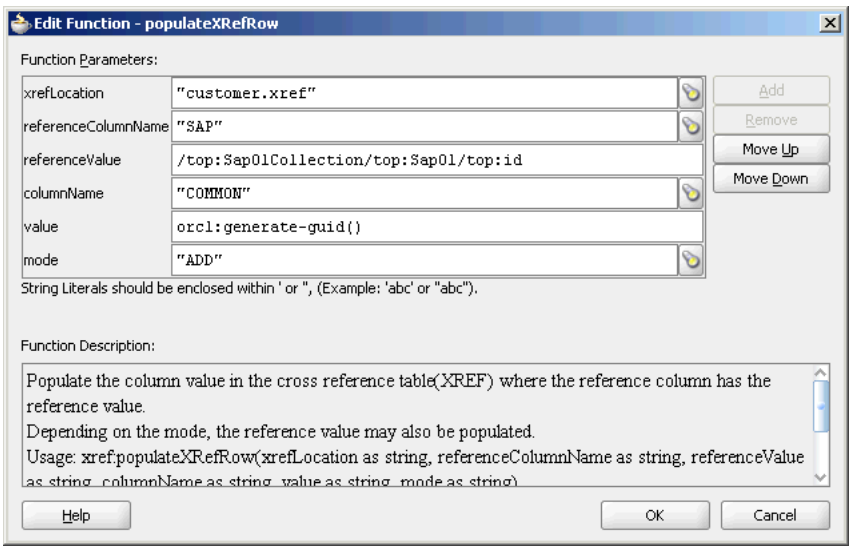
f. In the **mode** field, enter a mode in which you want to populate the cross reference table column. For example, enter `ADD`.

   You can also click **Browse** to select a mode. The Select Populate Mode dialog is displayed from which you can select a mode.

8. Click **OK**.

   A populated Edit Function – populateXRefRow dialog is shown in Figure 46–9.

Figure 46–9 Populated Edit Function – populateXRefRow Dialog



46.5 Looking Up Cross Reference Tables

After populating the cross reference table, you can use it to look up a value. The `xref:lookupXRef` and `xref:lookupXRef1M` functions enable you to look up a cross reference for single and multiple values, respectively.

46.5.1 About the xref:lookupXRef Function

You can use the `xref:lookupXRef` function to look up a cross reference column for a value that corresponds to a value in a reference column. For example, the following function looks up the `Common` column of the cross reference tables described in Table 46–2 for a value corresponding to the `SAP_001` value in the `SAP` column.

```
xref:lookupXRef("customers.xref","SAP","SAP_001","Common",true())
```

The syntax of the `xref:lookupXRef` function is shown in Example 46–4.

Example 46–4 xref:lookupXRef Function

```
xref:lookupXRef(xrefLocation as string, xrefReferenceColumnName as string,
xrefReferenceValue as string, xrefColumnName as string, needAnException as
boolean) as string
```

Parameters

- `xrefLocation`: The cross reference URI.

- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: When the value is set to `true`, an exception is thrown if the value is not found. Otherwise, an empty value is returned.

Exception Reasons

At runtime, an exception can occur for the following reasons:

- The cross reference table with the given name is not found.
- The specified column names are not found.
- The specified reference value is empty.
- Multiple values are found.

46.5.2 About the xref:lookupXRef1M Function

You can use the `xref:lookupXRef1M` function to look up a cross reference column for multiple values corresponding to a value in a reference column. The `xref:lookupXRef1M` function returns a node-set containing multiple nodes. Each node in the node-set contains a value.

For example, the following function looks up the `SAP` column of Table 46–11 for multiple values corresponding to the `EBS_1001` value in the `EBS` column:

```
xref:lookupXRef1M("customers.xref","EBS","EBS_1001","SAP",true())
```

The syntax of the `xref:lookupXRefRow1M` function is shown in Example 46–5.

Example 46–5 xref:lookupXRefRow1M Function

```
xref:lookupXRef1M(xrefLocation as String, xrefReferenceColumnName as String,
 xrefReferenceValue as String, xrefColumnName as String, needAnException as
 boolean) as node-set
```

Parameters

- `xrefLocation`: The cross reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: If this value is set to `true`, an exception is thrown when the referenced value is not found. Otherwise, an empty node-set is returned.

Example of the xref:lookupXRefRow1M Function

Consider the `Order` table shown in Table 46–14 with the following three columns: `Siebel`, `Billing1`, and `Billing2`.

Table 46–14 Order Table

Siebel	Billing1	Billing2
100	101	102

Table 46–14 (Cont.) Order Table

Siebel	Billing1	Billing2
110		111
		112

For 1:1 mapping, the `xref:lookupPopulatedColumns("Order","Siebel","100","false")` method returns the values shown in Example 46–6.

Example 46–6 xref:lookupPopulatedColumns Method

```
<column name="BILLING1">101</column>
<column name="BILLING2">102</column>
```

In this case, both the columns, `Billing1` and `Billing2`, are populated.

For 1:M mapping, the `xref:lookupPopulatedColumns("Order","Siebel","110","false")` method returns the values shown in Example 46–7.

Example 46–7 xref:lookupPopulatedColumns

```
<column name="BILLING2">111</column>
<column name="BILLING2">112</column>
```

In this case, `Billing1` is not populated.

Exception Reasons

An exception can occur for the following reasons:

- The cross reference table with the given name is not found.
- The specified column names are not found.
- The specified reference value is empty.

46.5.3 About the xref:lookupPopulatedColumns Function

You can use the `xref:lookupPopulatedColumns` function to look up all the populated columns for a given cross reference table, a cross reference column, and a value. The `xref:lookupPopulatedColumns` function returns a node-set with each node containing a column name and the corresponding value.

The syntax of the `xref:LookupPopulatedColumns` function is shown in Example 46–8.

Example 46–8 xref:LookupPopulatedColumns Function

```
xref:LookupPopulatedColumns(xrefTableName as String,xrefColumnName as
 String,xrefValue as String,needAnException as boolean)as node-set
```

Parameters

- `xrefTableName`: The name of the reference table.
- `xrefColumnName`: The name of the reference column.
- `xrefValue`: The value corresponding to the reference column name.

- `needAnException`: If this value is set to `true`, then an exception is thrown when no value is found in the referenced column. Otherwise, an empty node-set is returned.

Exception Reasons

An exception can occur for the following reasons:

- The cross reference table with the given name is not found.
- The specified column names are not found.
- The specified reference value is empty.

46.5.4 How to Look Up a Cross Reference Table for a Value

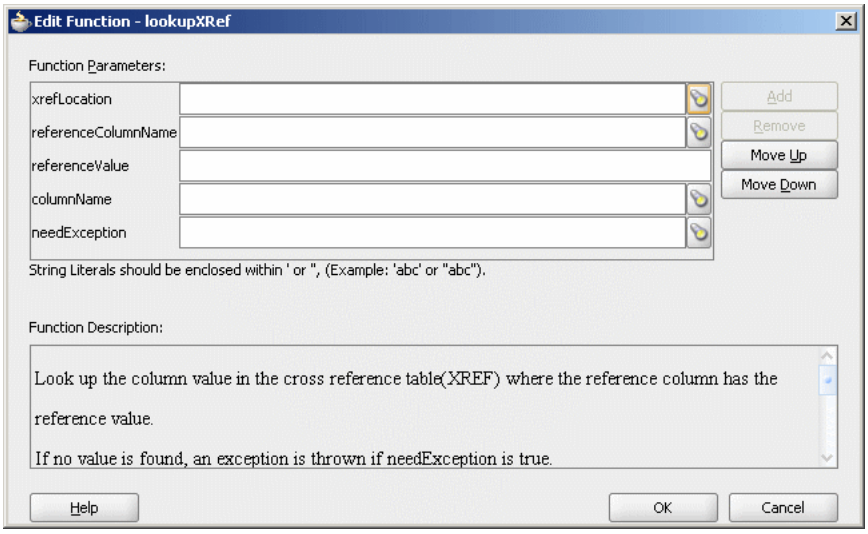
To look up a cross reference table column:

1. In the XSLT Mapper, expand the trees in the **Source** and **Target** panes.
2. Drag and drop the source element to the target element.
3. In the Component Palette, select **Advanced**.
4. Select **XREF Functions**.
5. Drag and drop the **lookupXRef** function to the line that connects the source object to the target object.

   A **lookupXRef** icon appears on the connecting line.
6. Double-click the **lookupXRef** icon.

   The Edit Function – lookupXRef dialog is displayed, as shown in Figure 46–10.

Figure 46–10 Edit Function – lookupXRef Dialog



7. Specify the following values for the fields in the Edit Function – lookupXRef dialog:

   a. In the **xrefLocation** field, enter the location URI of the cross reference file.

Click **Browse** to the right of the **xrefLocation** field to select the cross reference file. You can select an already deployed cross reference from MDS and also from a shared location in MDS by using the Resource Palette.

b. In the **referenceColumnName** field, enter the name of the cross reference column.

Click **Browse** to the right of the **referenceColumnName** field to select a column name from the columns defined for the cross reference you previously selected.

c. In the **referenceValue** field, you can manually enter a value or press **Ctrl-Space** to use the XPath Building Assistant. Press the up and down keys to locate an object in the list and press **Enter** to select that object.

d. In the **columnName** field, enter the name of the cross reference column.

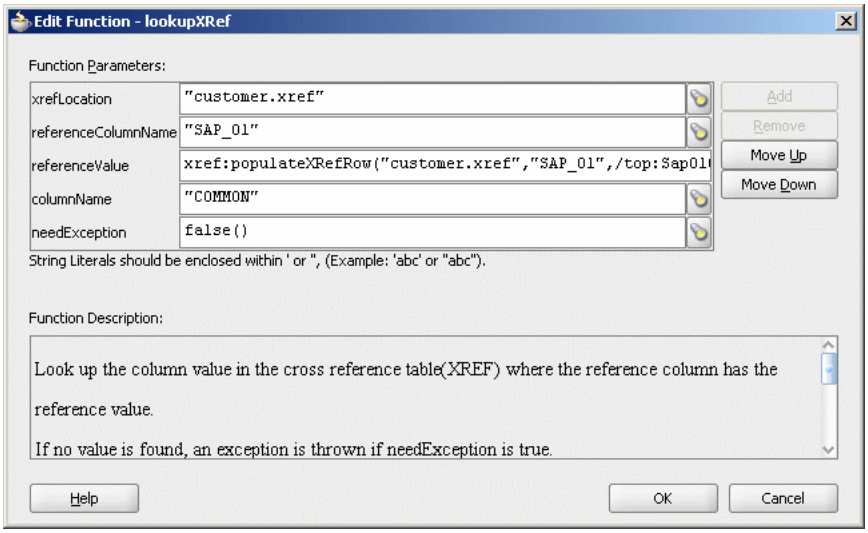
Click **Browse** to the right of the **columnName** field to select a column name from the columns defined for the cross reference you previously selected.

e. Click **Browse** to the right of **needException** field. The Need Exception dialog is displayed. Select **Yes** to raise an exception if no value is found. Otherwise, select **No**.

8. Click **OK**.

A populated Edit Function – lookupXRef dialog is shown in Figure 46–11.

Figure 46–11 Populated Edit Function – lookupXRef Dialog



46.6 Deleting a Cross Reference Table Value

You can use the `xref:markForDelete` function to delete a value in a cross reference table. The value in the column is marked as deleted. This function returns `true` if the deletion is successful. Otherwise, it returns `false`.

Any column value marked for deletion is treated as if the value does not exist. Therefore, you can populate the same column with the `xref:populateXRefRow` function in `ADD` mode.

Note: Using a column value marked for deletion as a reference value in `LINK` mode of the `xref:populateXRefRow` function raises an error.

A cross reference table row should have at least two mappings. If you have only two mappings in a row and you mark one value for deletion, then the value in another column is also deleted.

The syntax for the `xref:markForDelete` function is shown in Example 46–9.

Example 46–9 xref:markForDelete Function

```
xref:markForDelete(xrefTableName as string, xrefColumnName as string,
xrefValueToDelete as string) return as boolean
```

Parameters

- `xrefTableName`: The cross reference table name.
- `xrefColumnName`: The name of the column from which you want to delete a value.
- `xrefValueToDelete`: The value to be deleted.

Exception Reasons

An exception can occur for the following reasons:

- The cross reference table with the given name is not found.
- The specified column name is not found.
- The specified value is empty.
- The specified value is not found in the column.
- Multiple values are found.

46.6.1 How to Delete a Cross Reference Table Value

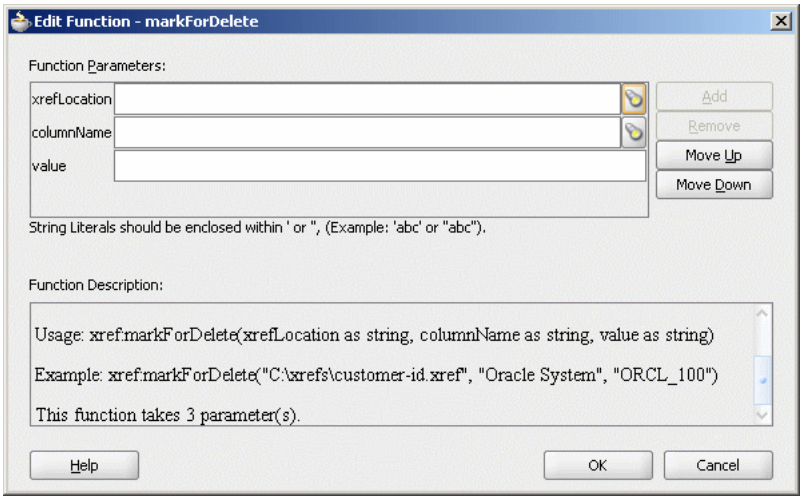
To delete a cross reference table value:

1. In the XSLT Mapper, expand the trees in the **Source** and **Target** panes.
2. Drag and drop the source element to the target element.
3. In the Component Palette, select **Advanced**.
4. Select **XREF Functions**.
5. Drag and drop the **markForDelete** function to the line that connects the source object to the target object.

   A **markForDelete** icon appears on the connecting line.
6. Double-click the **markForDelete** icon.

   The Edit Function – markForDelete dialog is displayed, as shown in Figure 46–12.

Figure 46–12 Edit Function – markForDelete Dialog



7. Specify the following values for the fields in the Edit Function – markForDelete dialog:

   a. In the **xrefLocation** field, enter the location URI of the cross reference file.

      Click the **Search** icon to the right of the **xrefLocation** field to select the cross reference file. You can select an already deployed cross reference from MDS and also from a shared location in MDS by using the Resource Palette.

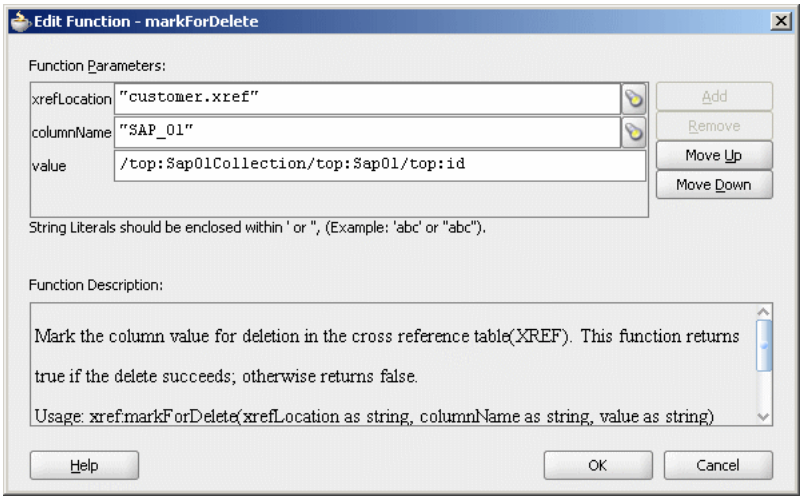
   b. In the **columnName** field, enter the name of cross reference table column.

      Click the **Search** icon to the right of the **columnName** field to select a column name from the columns defined for the cross reference you previously selected.

   c. In the **Value** field, manually enter a value or press **Ctrl-Space** to launch the XPath Building Assistant. Press the up and down keys to locate an object in the list and press **Enter** to select that object.

   A populated Edit Function – markForDelete dialog is shown in Figure 46–13.

Figure 46–13 Populated Edit Function – markForDelete Dialog

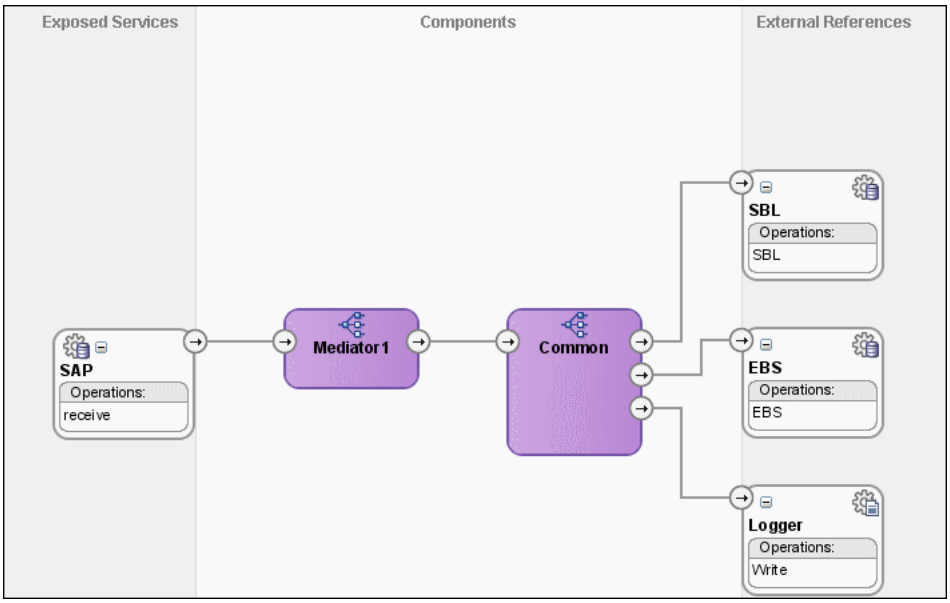


8. Click **OK**.

46.7 Creating and Running the Cross Reference Use Case

This cross reference use case implements an integration scenario between Oracle EBS, SAP, and Siebel instances. In this use case, when an insert, update, or delete operation is performed on the `SAP_01` table, the corresponding data is inserted or updated in the `EBS` and `SBL` tables. Figure 46–14 provides an overview of this use case.

Figure 46–14 XrefCustApp Use Case in SOA Composite Editor



For downloading the sample files mentioned in this section, visit the following URL:

`https://soasamples.samplecode.oracle.com/#mediator`

46.7.1 How to Create the Use Case

This section provides the design-time tasks for creating, building, and deploying your SOA Composite application. These tasks should be performed in the order in which they are presented.

46.7.1.1 Task 1: How to Configure the Oracle Database and Database Adapter

To configure the Oracle database and database adapter:

1. You need the `SCOTT` database account with password `TIGER` for this use case. You must ensure that the `SCOTT` account is unlocked.

   You can log in as `SYSDBA` and then run the `setup_user.sql` script available in the `XrefOrderApp1M/sql` directory to unlock the account.

2. Run the `create_schema.sql` script available in the `XrefOrderApp1M/sql` directory to create the tables required for this use case.

3. Run the `create_app_procedure.sql` script available in the `XrefOrderApp1M/sql` directory to create a procedure that simulates the various applications participating in this integration.

4. Run the `createschema_xref_oracle.sql` script available in the `OH/rcu/integration/soainfra/sql/xref/` directory to create a cross reference table to store runtime cross reference data.

5. Copy the `ra.xml` and `weblogic-ra.xml` files from `$BEAHOME/META-INF` to the newly created directory called `META-INF` on your computer.
6. Edit the `weblogic-ra.xml` file available in the `$BEAHOME/META-INF` directory as follows:
   - Modify the property to `xADataSourceName` as follows:

     ```
     <property>
      <name>xADataSourceName</name>
      <value>jdbc/DBConnection1</value>
     </property>
     ```

   - Modify the `jndi-name` as follows:

     ```
     <jndi-name> eis/DB/DBConnection1</jndi-name>
     ```

   This sample uses `eis/DB/DBConnection1` to poll the SAP table for new messages and to connect to the procedure that simulates Oracle EBS and Siebel instances.
7. Package the `ra.xml` and `weblogic-ra.xml` files as a RAR file and deploy the RAR file by using Oracle WebLogic Server Administration Console.
8. Create a data source using the Oracle WebLogic Server Administration Console with the following values:
   - **jndi-name**=`jdbc/DBConnection1`
   - **user**=`scott`
   - **password**=`tiger`
   - **url**=`jdbc:oracle:thin:@host:port:service`
   - **connection-factory**
     factory-class=`oracle.jdbc.pool.OracleDataSource`
9. Create a data source using the Oracle WebLogic Server Administration Console with the following values:
   - **jndi-name**=`jdbc/xref`
   - **user**=`scott`
   - **password**=`tiger`
   - **url**=`jdbc:oracle:thin:@host:port:service`
   - **connection-factory**
     factory-class=`oracle.jdbc.pool.OracleDataSource`

46.7.1.2 Task 2: 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 `XrefCustApp`, and then click **Next**.

   The Name your SOA project page appears.

5. In the **Project Name** field, enter `XrefCustApp` 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 updated with the new application and project and the SOA Composite Editor contains a blank composite.

7. From the **File** menu, select **Save All**.

46.7.1.3 Task 3: How to Create a Cross Reference

After creating an application and a project for the use case, you must create a cross reference table.

To create a cross reference table:

1. In the Application Navigator, right-click the **XrefCustApp** project and select **New**.
2. In the New Gallery dialog, expand the **SOA Tier** node, and then select the **Transformations** category.
3. In the **Items** list, select **Cross Reference(XREF)** and click **OK**.

   The Create Cross Reference(XREF) File dialog is displayed.

4. In the **File Name** field, enter `customer.xref`.
5. In the **End System** fields, enter `SAP_01` and `EBS_i76`.
6. Click **OK**.

   The Cross Reference Editor is displayed.

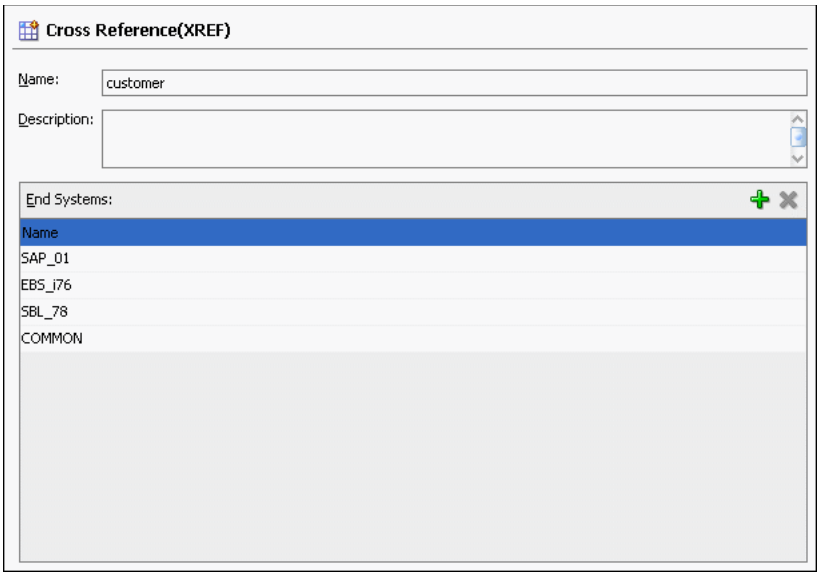
7. Click **Add**.

   A new row is added.

8. Enter `SBL_78` as the end system name in the newly added row.
9. Click **Add** and enter `Common` as the end system name.

   The Cross Reference Editor appears, as shown in Figure 46–15.

Figure 46–15 Customer Cross Reference



10. From the **File** menu, select **Save All** and close the Cross Reference Editor.

46.7.1.4 Task 4: How to Create a Database Adapter Service

To create a database adapter service:

1. In the Component Palette, select **SOA**.
2. Select **Database Adapter** and drag it to the **Exposed Services** 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 `SAP`.
5. Click **Next**.

   The Service Connection page is displayed.
6. In the **Application Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter `eis/DB/DBConnection1`.
8. Click **Next**.

   The Operation Type page is displayed.
9. Select **Poll for New or Changed Records in a Table** and click **Next**.

   The Select Table page is displayed.
10. Click **Import Tables**.

    The Import Tables dialog is displayed.
11. Select **Scott** from **Schema**.
12. In the **Name Filter** field, enter `%SAP%` and click **Query**.

    The **Available** field is populated with `SAP_01` table name.
13. Double-click **SAP_01**.

The **selected** field is populated with **SAP_01**.

14. Click **OK**.

    The Select Table page now contains the **SAP_01** table.

15. Select **SAP_01** and click **Next**.

    The Define Primary Key page is displayed.

16. Select **ID** as the primary key and click **Next**.

    The Relationships page is displayed.

17. Click **Next**.

    The Attribute Filtering page is displayed.

18. Click **Next**.

    The After Read page is displayed.

19. Select **Update a Field in the [SAP_01] Table (Logical Delete)** and click **Next**.

    The Logical Delete page is displayed.

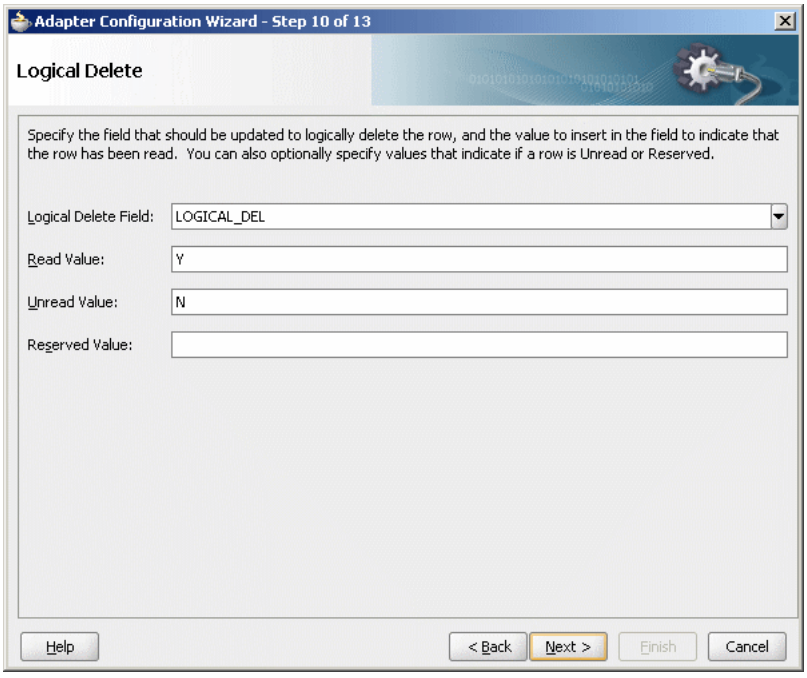
20. In the **Logical Delete** field, select **LOGICAL_DEL**.

21. In the **Read Value** field, enter `Y`.

22. In the **Unread Value** field, enter `N`.

    Figure 46–16 shows the Logical Delete page of the Adapter Configuration wizard.

Figure 46–16 Logical Delete Page: Adapter Configuration Wizard



23. Click **Next**.

    The Polling Options page is displayed.

24. Click **Next**.

    The Define Selection Criteria page is displayed.

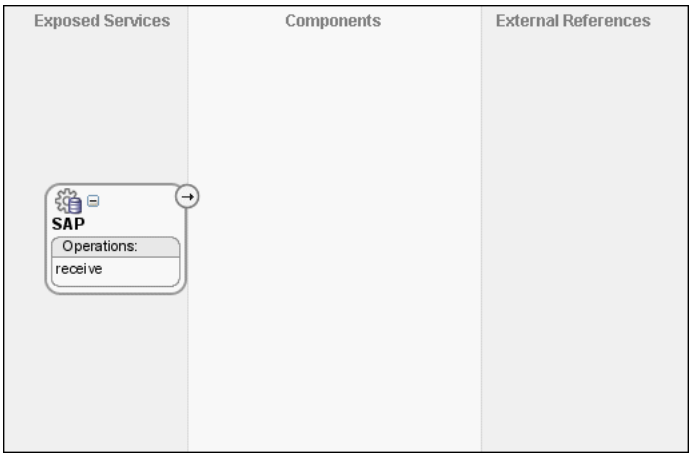
25. Click **Next**.

    The Finish page is displayed.

26. Click **Finish**.

    A database adapter service named **SAP** is created, as shown in Figure 46–17.

Figure 46–17 SAP Database Adapter Service in SOA Composite Editor



27. From the **File** menu, select **Save All**.

46.7.1.5 Task 5: How to Create EBS and SBL External References

To create EBS and SBL external references:

1. In the Component Palette, select **SOA**.
2. Select **Database 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 `EBS`.
5. Click **Next**.

   The Service Connection page is displayed.

6. In the **Application Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter `eis/DB/DBConnection1`.
8. Click **Next**.

   The Operation Type page is displayed.

9. Select **Call a Stored Procedure or Function** and click **Next**.

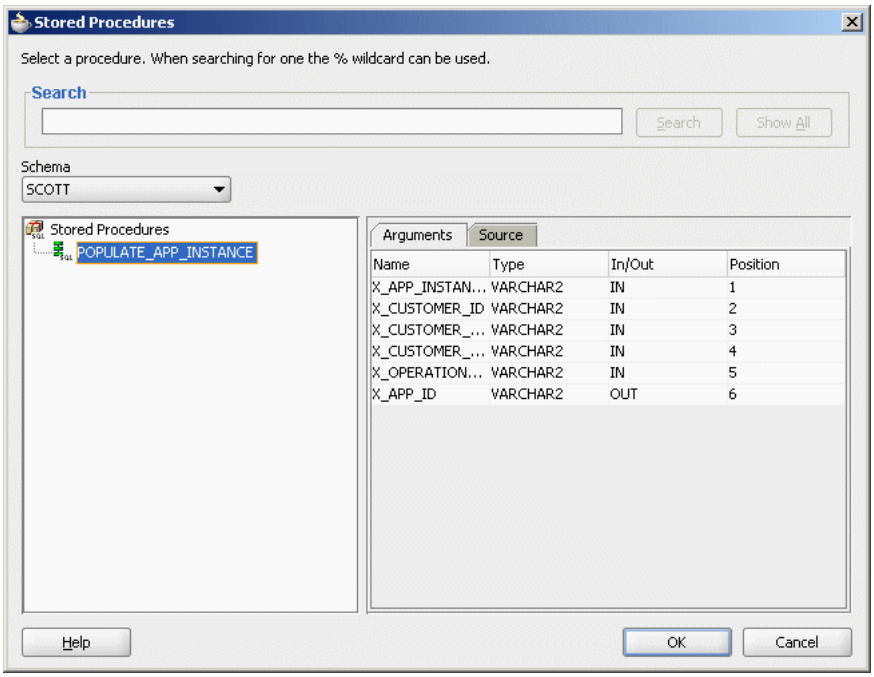
   The Specify Stored Procedure page is displayed.

10. Select **Scott** from **Schema**.
11. Click **Browse**.

    The Stored Procedures dialog is displayed.

12. Select **POPULATE_APP_INSTANCE**, as shown in Figure 46–18.

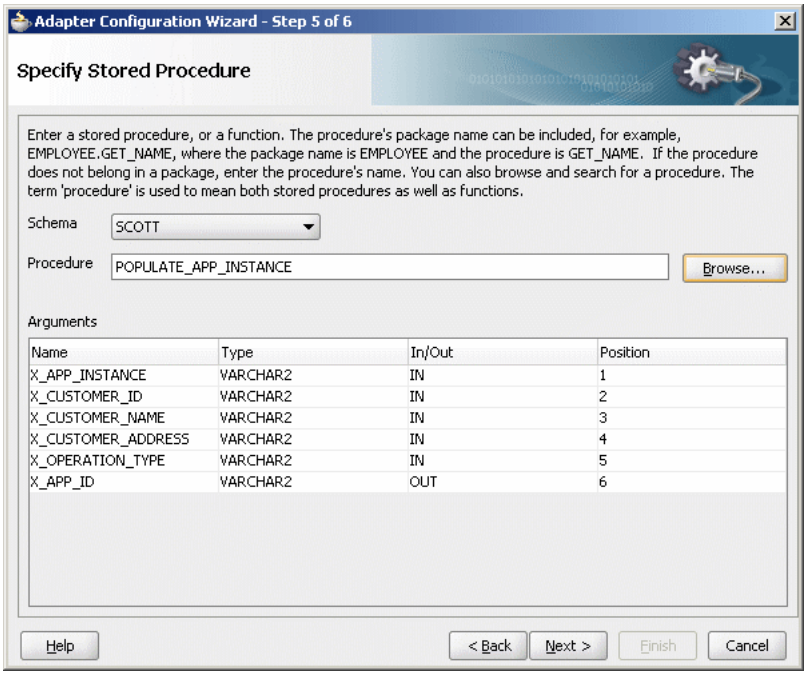
Figure 46–18 Stored Procedure Dialog



13. Click **OK**.

    The Specify Stored Procedure page appears, as shown in Figure 46–19.

Figure 46–19 Specify Stored Procedure Page of Adapter Configuration Wizard



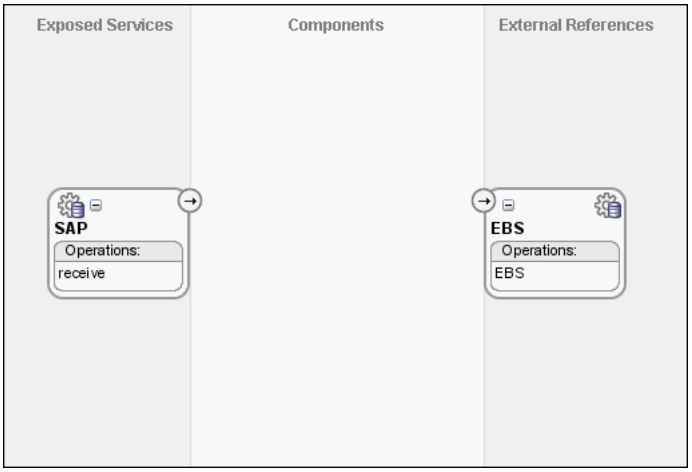
14. Click **Next**.

The Finish page is displayed.

15. Click **Finish**.

    Figure 46–20 shows the **EBS** reference in the SOA Composite Editor.

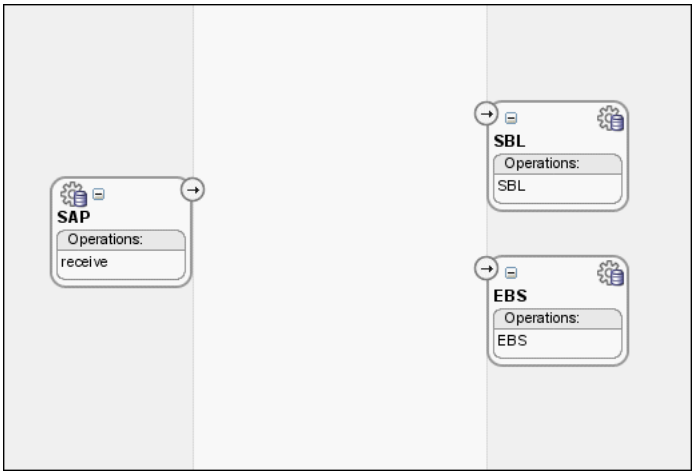
Figure 46–20 EBS Reference in SOA Composite Editor



16. From the **File** menu, select **Save All**.
17. Repeat Step 2 through Step 16 to create another external reference named `SBL`.

    After completing this task, the SOA Composite Editor appears, as shown in Figure 46–21.

Figure 46–21 SBL Reference in SOA Composite Editor



46.7.1.6 Task 6: How to Create the Logger File Adapter External Reference

To create the Logger file adapter external 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 `Logger`.
5. Click **Next**.

   The Operation page is displayed.
6. In the **Operation Type** field, select **Write File**.
7. Click **Next**.

   The File Configuration page is displayed.
8. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory in which you want to write the files.
9. In the **File Naming Convention** field, enter `output.xml` and click **Next**.

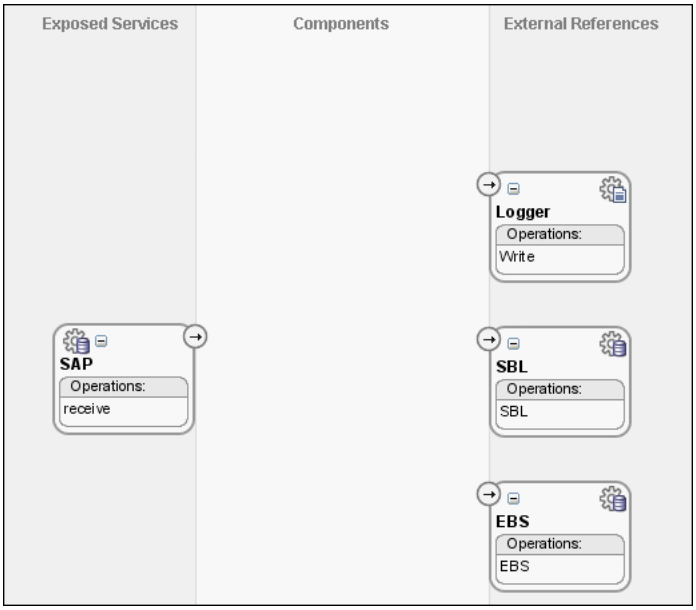
   The Messages page is displayed.
10. Click **Search**.

    The Type Chooser dialog is displayed.
11. Navigate to **Type Explorer > Project Schema Files > SCOTT_POPULATE_APP_INSTANCE.xsd**, and then select **OutputParameters**.
12. Click **OK**.
13. Click **Next.**

    The Finish page is displayed.
14. Click **Finish**.

    Figure 46–22 shows the **Logger** reference in the SOA Composite Editor.

Figure 46–22 Logger Reference in SOA Composite Editor



15. From the **File** menu. select **Save All**.

46.7.1.7 Task 7: How to Create an Oracle Mediator Service Component

To create an Oracle Mediator service component:

1. Drag and drop a **Mediator** icon from the Component Palette to the **Components** section of the SOA Composite Editor.

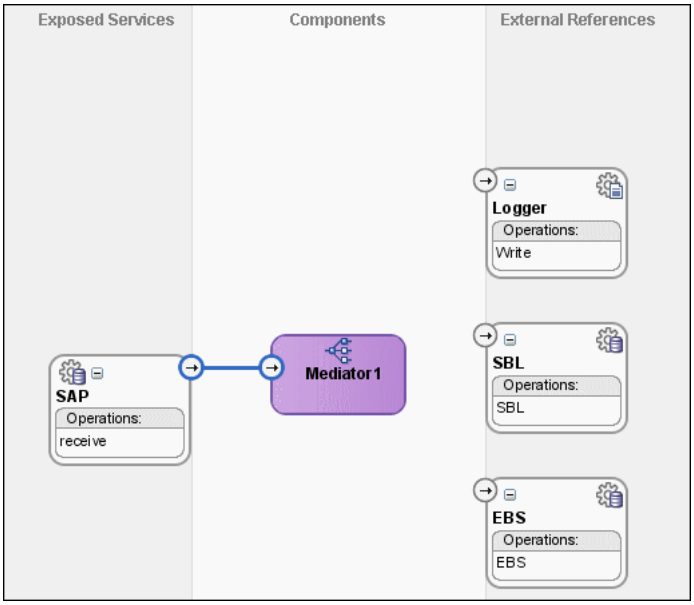
   The Create Mediator dialog is displayed.

2. From the **Template** list, select **Define Interface Later**.
3. Click **OK**.

   An Oracle Mediator with name `Mediator1` is created.

4. Connect the **SAP** service to the **Mediator1**, as shown in Figure 46–23.

Figure 46–23 SAP Service Connected to Mediator1



5. From the **File** menu, select **Save All**.
6. Drag and drop another **Mediator** icon from the Component Palette to the **Components** section of the SOA Composite Editor.

   The Create Mediator dialog is displayed.

7. From the **Template** list, select **Interface Definition From WSDL**.
8. Deselect **Create Composite Service with SOAP Bindings**.
9. To the right of the **WSDL File** field, click **Find Existing WSDLs**.
10. Navigate to and then select the **Common.wsdl** file. The **Common.wsdl** file is available in the **Samples** folder.
11. Click **OK**.
12. Click **OK**.

    An Oracle Mediator with name **Common** is created.

46.7.1.8 Task 8: How to Specify Routing Rules for an Oracle Mediator Service Component

You must specify routing rules for the following operations:

- Insert
- Update
- UpdateID
- Delete

To create routing rules for an insert operation:

1. Double-click the **Mediator1** Oracle Mediator.

   The Mediator Editor is displayed.

2. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.

3. Select **Service**.

   The Target Services dialog is displayed.

4. Navigate to **XrefCustApp** > **Mediators** > **Common**, **Services** > **Common**.

5. Select **Insert** and click **OK**.

6. Click the **Filter** icon.

   The Expression Builder dialog is displayed.

7. In the **Expression** field, enter the following expression:

   ```
   $in.Sap01Collection/top:Sap01Collection/top:Sap01/top:operation='INSERT'
   ```

8. Click **OK**.

9. Next to the **Transform Using** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.

10. Select **Create New Mapper File** and enter `SAP_TO_COMMON_INSERT.xsl`.

11. Click **OK**.

    An **SAP_TO_COMMON_INSERT.xsl** file is displayed in the XSLT Mapper.

12. Drag and drop the **top:SAP01** source element to the **inp1:Customer** target element.

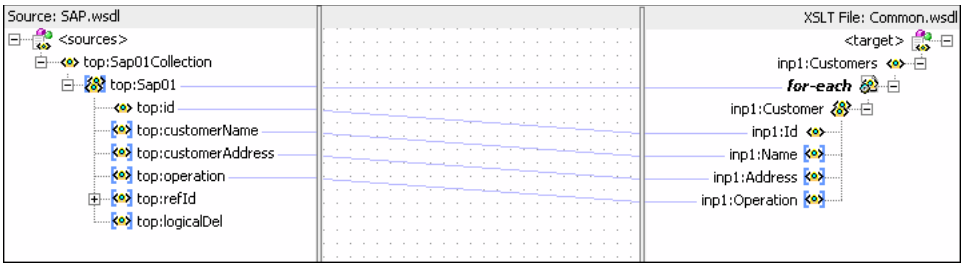
    The Auto Map Preferences dialog is displayed.

13. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names**.

14. Click **OK**.

    The transformation is created, as shown in Figure 46–24.

Figure 46–24 SAP_TO_COMMON_INSERT.xsl Transformation



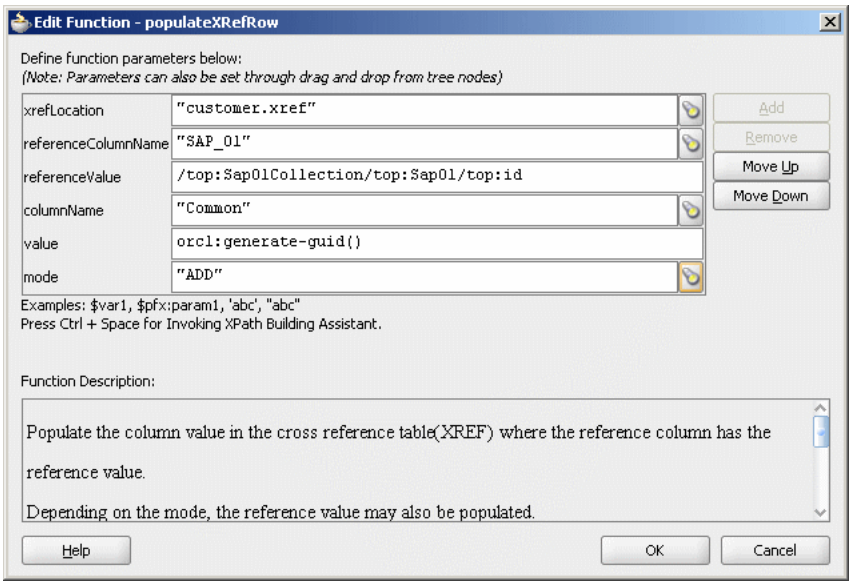
15. From the Component Palette, select **Advanced**.
16. Select **XREF Functions**.
17. Drag and drop the **populateXRefRow** function from the Component Palette to the line connecting the **top:id** and **inp1:id** elements.
18. Double-click the **populateXRefRow** icon.

    The Edit Function-populateXRefRow dialog is displayed.
19. Click **Search** to the right of the **xrefLocation** field.

    The SOA Resource Lookup dialog is displayed.
20. Select **customer.xref** and click **OK**.
21. In the **referenceColumnName** field, enter `"SAP_01"` or click **Search** to select the column name.
22. In the **referenceValue** column, enter `/top:Sap01Collection/top:Sap01/top:id`.
23. In the **columnName** field, enter `"Common"` or click **Search** to select the column name.
24. In the **value** field, enter `oraext:generate-guid()`.
25. In the **mode** field, enter `"Add"` or click **Search** to select this mode.

    Figure 46–25 shows the populated Edit Function – populateXRefRow dialog.

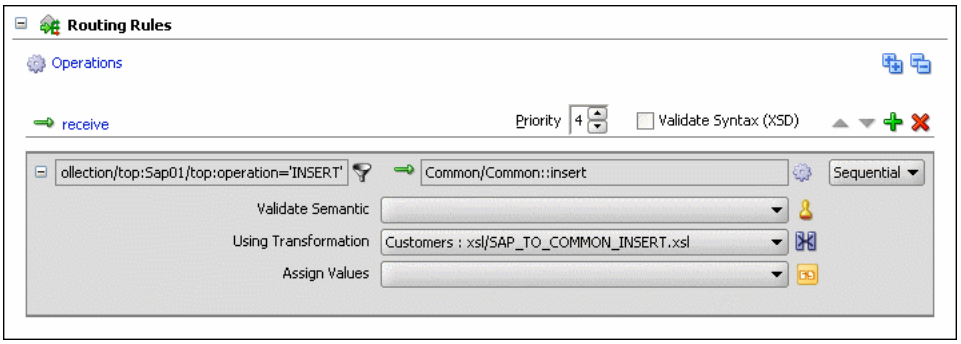
Figure 46–25 Edit Function – populateXRefRow Dialog: XrefCustApp Use Case



26. Click **OK**.
27. From the **File** menu, select **Save All** and close the **SAP_TO_COMMON_INSERT.xsl** file.

    The **Routing Rules** section appears, as shown in Figure 46–26.

Figure 46–26 Routing Rules Section with Insert Operation



To create routing rules for an update operation:

1. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.
2. Select **Service**.

   The Target Services dialog is displayed.
3. Navigate to **XrefCustApp** > **Mediators** > **Common**, **Services** > **Common**.
4. Select **Update** and click **OK**.
5. Click the **Filter** icon.

   The Expression Builder dialog is displayed.
6. In the **Expression** field, enter the following expression:

```
$in.Sap01Collection/top:Sap01Collection/top:Sap01/top:operation='UPDATE'
```

7. Click **OK**.
8. Next to the **Transform Using** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.
9. Select **Create New Mapper File** and enter `SAP_TO_COMMON_UPDATE.xsl`.
10. Click **OK**.

    An **SAP_TO_COMMON_UPDATE.xsl** file is displayed.
11. Drag and drop the **top:Sap01** source element to the **inp1:Customer** target element.

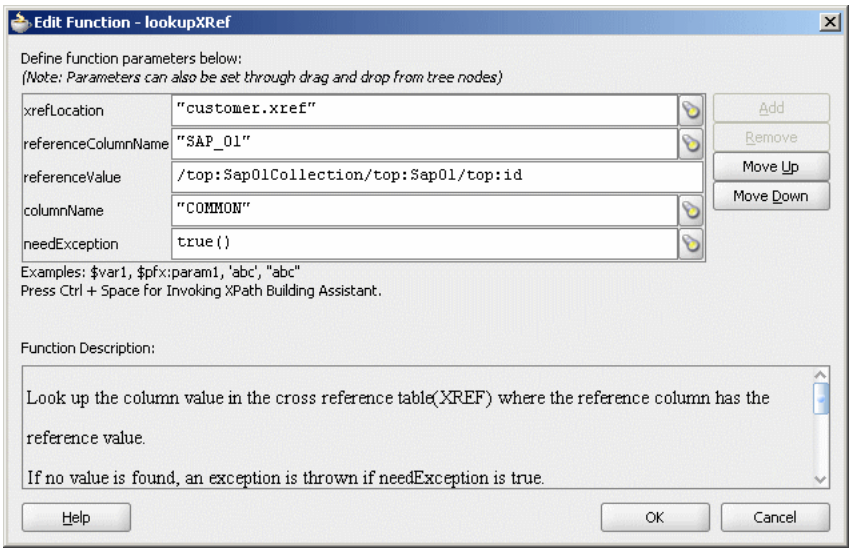
    The Auto Map Preferences dialog is displayed.
12. Click **OK**.
13. From the Component Palette, select **Advanced**.
14. Select **XREF Functions**.
15. Drag and drop the **lookupXRef** function from the Component Palette to the line connecting the **top:id** and **inp1:id** elements.
16. Double-click the **lookupXRef** icon.

    The Edit Function-lookupXRef dialog is displayed.
17. To the right of the **xrefLocation** field, click **Search**.

    The SOA Resource Lookup dialog is displayed.
18. Select **customer.xref** and click **OK**.
19. In the **referenceColumnName** field, enter `"SAP_01"` or click **Search** to select the column name.
20. In the **referenceValue** column, enter `/top:Sap01Collection/top:Sap01/top:id`.
21. In the **columnName** field, enter `"COMMON"` or click **Search** to select the column name.
22. In the **needException** field, enter `true()` or click **Search** to select this mode.

    Figure 46–27 shows the populated Edit Function – looupXRef dialog.

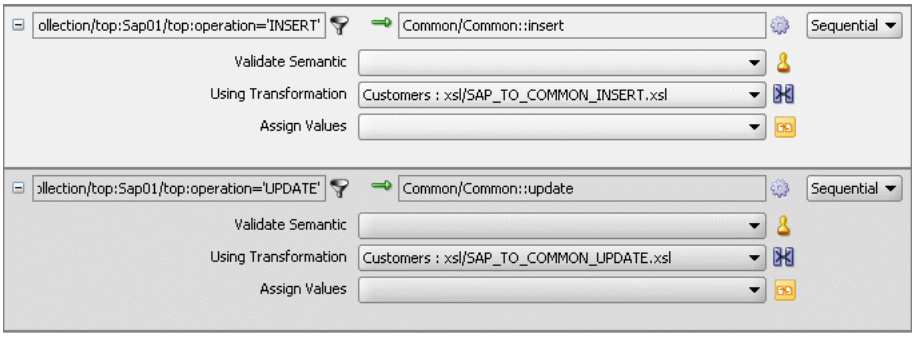
Figure 46–27 Edit Function – lookupXRef Dialog: XrefCustApp Use Case



23. Click **OK**.
24. From the **File** menu, select **Save All** and close the **SAP_TO_COMMON_UPDATE.xsl** file.

    The **Routing Rules** section appears, as shown in Figure 46–28.

Figure 46–28 Insert Operation and Update Operation



To create routing rules for an updateID operation:

Perform the following tasks to create routing rules for an `updateID` operation:

1. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.
2. Select **Service**.

   The Target Services dialog is displayed.
3. Navigate to **XrefCustApp** > **Mediators** > **Common**, **Services** > **Common**.
4. Select **updateid** and click **OK**.
5. Click the **Filter** icon.

   The Expression Builder dialog is displayed.
6. In the **Expression** field, enter the following expression:

```
$in.Sap01Collection/top:Sap01Collection/top:Sap01/top:operation = 'UPDATEID'
```

7. Click **OK**.
8. Next to the **Transform Using** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.
9. Select **Create New Mapper File** and enter `SAP_TO_COMMON_UPDATEID`.xsl.
10. Click **OK**.

    An **SAP_TO_COMMON_UPDATEID.xsl** file is displayed.
11. Drag and drop the **top:Sap01** source element to the **inp1:Customer** target element.

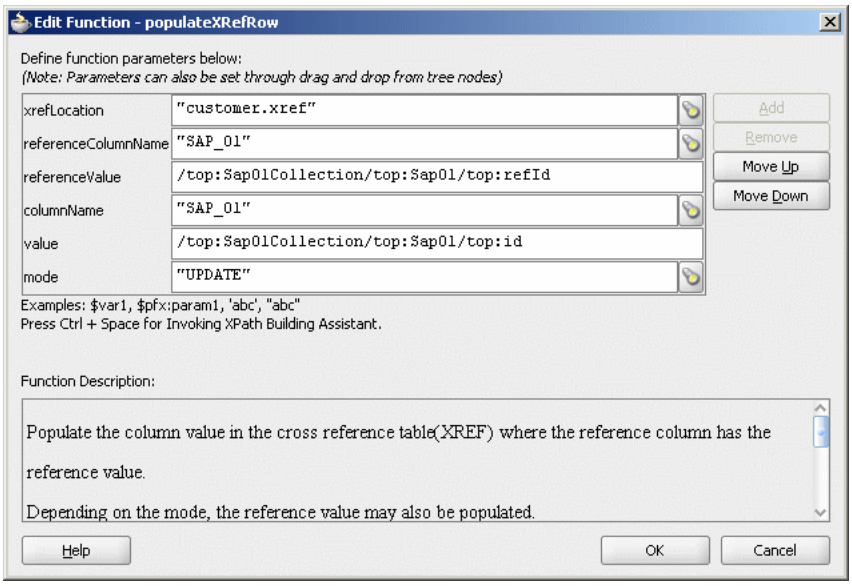
    The Auto Map Preferences dialog is displayed.
12. Click **OK**.
13. From the Component Palette, select **Advanced**.
14. Select **XREF Functions**.
15. Drag and drop the **populateXRefRow** function from the Component Palette to the line connecting the **top:id** and **inp1:id** elements.
16. Double-click the **populateXRefRow** icon.

    The Edit Function-populateXRefRow dialog is displayed.
17. To the right of the **xrefLocation** field, click **Search**.

    The SOA Resource Lookup dialog is displayed.
18. Select **customer.xref** and click **OK**.
19. In the **referenceColumnName** field, enter `"SAP_01"` or click **Search** to select the column name.
20. In the **referenceValue** column, enter `/top:Sap01Collection/top:Sap01/top:refId`.
21. In the **columnName** field, enter `"SAP_01"` or click **Search** to select the column name.
22. In the **value** field, enter `/top:Sap01Collection/top:Sap01/top:Id`.
23. In the **mode** field, enter `"UPDATE"` or click **Search** to select this mode.

    Figure 46–29 shows a populated Edit Function – populateXRefRow dialog.

Figure 46–29 Edit Function – populateXRefRow Dialog: XrefCustApp Use Case



24. Drag and drop the **lookupXRef** function from the Component Palette to the line connecting the **top:id** and **inp1:id** elements.
25. Double-click the **lookupXRef** icon.

    The Edit Function-lookupXRef dialog is displayed.
26. To the right of the **xrefLocation** field, click **Search**.

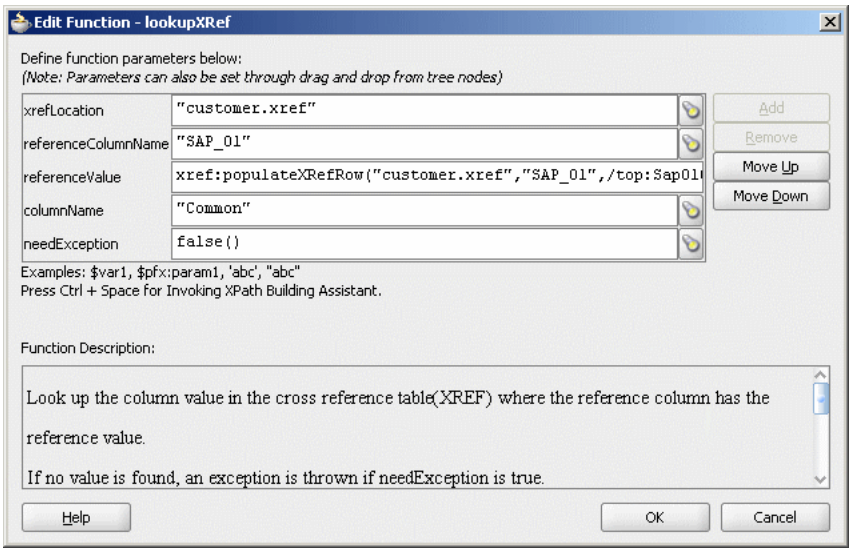
    The SOA Resource Lookup dialog is displayed.
27. Select **customer.xref** and click **OK**.
28. In the **referenceColumnName** field, enter `"SAP_01"` or click **Search** to select the column name.
29. In the **referenceValue** column, enter the following:

```
xref:populateXRefRow("customer.xref","SAP_
01",/top:Sap01Collection/top:Sap01/top:refId,"SAP_
01",/top:Sap01Collection/top:Sap01/top:id,"UPDATE").
```

30. In the **columnName** field, enter `"COMMON"` or click **Search** to select the column name.
31. In the **needException** field, enter `false()` or click **Search** to select this mode.

    Figure 46–30 shows a populated Edit Function – lookupXRef dialog.

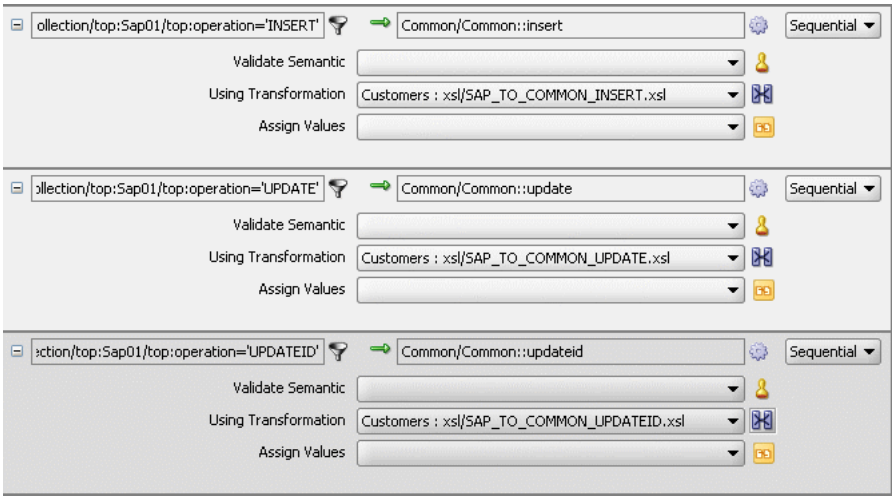
Figure 46–30 Edit Function – lookupXRef Dialog: XrefCustApp Use Case



32. Click **OK**.
33. From the **File** menu, select **Save All** and close the **SAP_TO_COMMON_UPDATEID.xsl** file.

    The **Routing Rules** section appears, as shown in Figure 46–31.

Figure 46–31 Insert, Update, and UpdateID Operations



To create routing rules for a delete operation:

1. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.
2. Select **Service**.

   The Target Services dialog is displayed.
3. Navigate to **XrefCustApp** > **Mediators** > **Common**, **Services** > **Common**.
4. Select **delete** and click **OK**.
5. Click the **Filter** icon.

The Expression Builder dialog is displayed.

6. In the **Expression** field, enter the following expression:

```
$in.Sap01Collection/top:Sap01Collection/top:Sap01/top:operation = 'DELETE'
```

7. Click **OK**.
8. Next to the **Transform Using** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.
9. Select **Create New Mapper File** and enter `SAP_TO_COMMON_DELETE.xsl`.
10. Click **OK**.

    A **SAP_TO_COMMON_DELETE.xsl** file is displayed.
11. Right-click **<sources>** and select **Add Parameter**.

    The Add Parameter dialog is displayed.
12. In the **Local Name** field, enter `COMMONID`.
13. Select **Set Default Value**.
14. Select **Expression**.
15. In the **XPath Expression** field, enter

```
xref:lookupXRef("customer.xref","SAP_
01",/top:Sap01Collection/top:Sap01/top:id,"COMMON",false()).
```

16. Click **OK**.
17. Drag and drop the **top:Sap01** source element to the **inp1:Customer** target element.

    The Auto Map Preferences dialog is displayed.
18. Click **OK**.
19. Delete the line connecting **top:id** and **inp1:id**.
20. Connect **COMMONID** to **inp1:id**.
21. Right-click **inp1:id** and select **Add XSL node** and then **if**.

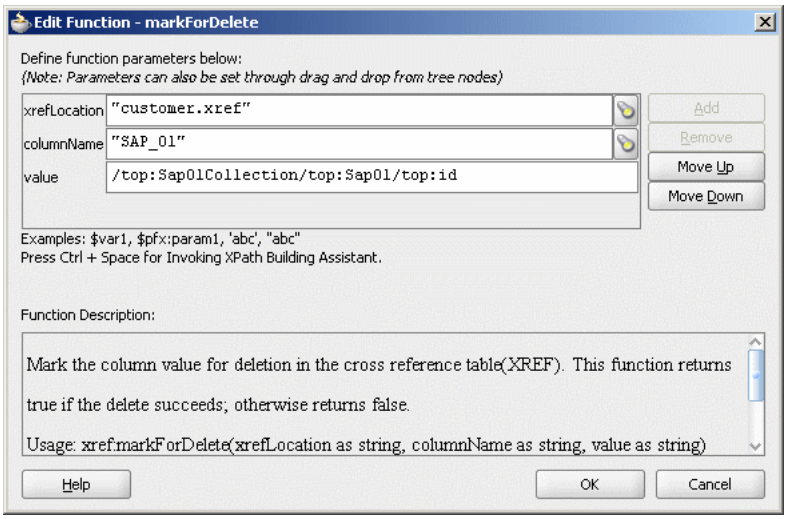
    A new node `if` is inserted between `inp1:customer` and `inp1:id`.
22. Connect **top:id** to the **if** node.
23. From the Component Palette, select **Advanced**.
24. Select **XREF Functions**.
25. Drag and drop the **markForDelete** function from the Component Palette to the line connecting **top:id** and the **if** node.
26. Double-click the **markForDelete** icon.

    The Edit Function-markForDelete dialog is displayed.
27. Click **Search** to the right of the **xrefLocation** field.

    The SOA Resource Lookup dialog is displayed.
28. Select **customer.xref** and click **OK**.
29. In the **columnName** field, enter `"SAP_01"` or click **Search** to select the column name.
30. In the **value** field, enter `/top:Sap01Collection/top:Sap01/top:Id`.

Figure 46–32 shows a populated Edit Function – markForDelete dialog.

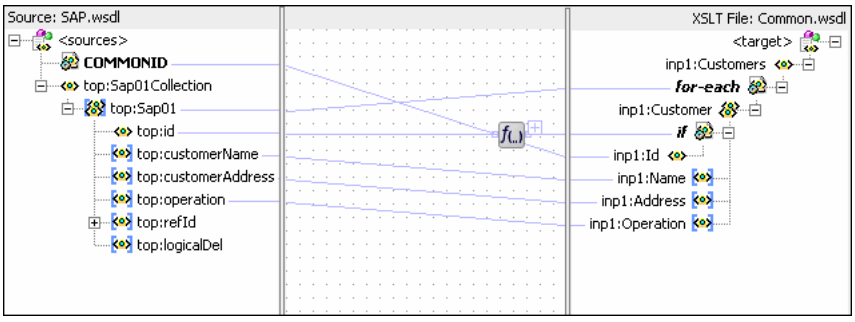
Figure 46–32 Edit Function – markForDelete Dialog: XrefCustApp Use Case



31. Click **OK**.

The **SAP_TO_COMMON_DELETE.xsl** file appears, as shown in Figure 46–33.

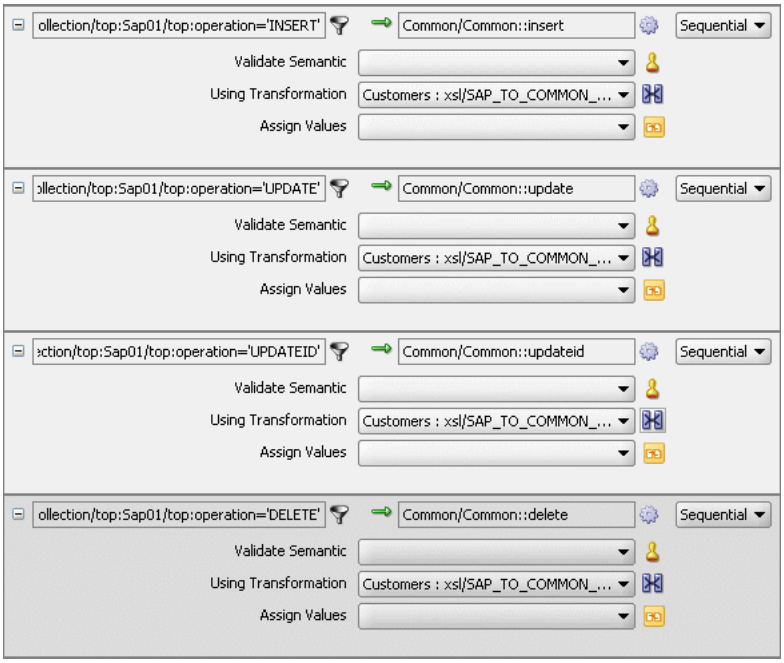
Figure 46–33 SAP_TO_COMMON_DELETE.xsl



32. From the **File** menu, select **Save All** and close the **SAP_TO_COMMON_DELETE.xsl** file.

The **Routing Rules** section appears, as shown in Figure 46–34.

Figure 46–34 Insert, Update, UpdateID, and Delete Operations



46.7.1.9 Task 9: How to Specify Routing Rules for the Common Oracle Mediator

You must specify routing rules for the following operations of the Common Oracle Mediator:

- Insert
- Delete
- Update
- UpdateID

To create routing rules for the insert operation:

1. Double-click the **Common** Oracle Mediator.

   The Mediator Editor is displayed.

2. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.

3. Select **Service**.

   The Target Services dialog is displayed.

4. Navigate to **XrefCustApp** > **References** > **SBL**.

5. Select **SBL** and click **OK**.

6. Next to the **Transform Using** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.

7. Select **Create New Mapper File** and enter `COMMON_TO_SBL_INSERT.xsl`.

8. Click **OK**.

   A **COMMON_TO_SBL_INSERT.xsl** file is displayed.

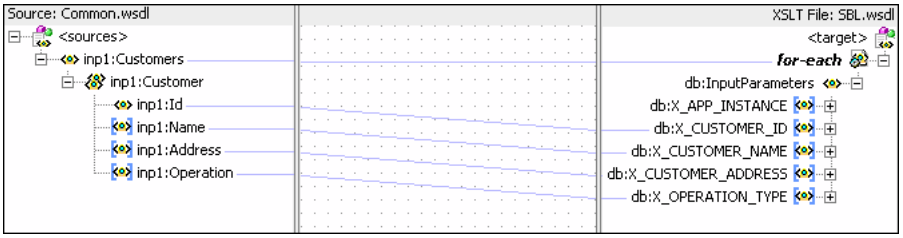
9. Drag and drop the **inp1:Customers** source element to the **db:InputParameters** target element.

   The Auto Map Preferences dialog is displayed.

10. Click **OK**.

    The transformation is created, as shown in Figure 46–35.

Figure 46–35 COMMON_TO_SBL_INSERT.xsl Transformation



11. From the **File** menu, select **Save All** and close the **COMMON_TO_SBL_INSERT.xsl** file.

12. In the **Synchronous Reply** section, click **Browse for target service operations**.

    The Target Type dialog is displayed.

13. Select **Service**.

    The Target Services dialog is displayed.

14. Navigate to **XrefCustApp** > **References** > **Logger**.

15. Select **Write** and click **OK**.

16. Next to the **Transform Using** field, click the **Transformation** icon.

    The Reply Transformation map dialog is displayed.

17. Select **Create New Mapper File** and enter `SBL_TO_COMMON_INSERT.xsl`.

18. Select **Include Request in the Reply Payload**.

19. Click **OK**.

    A **SBL_TO_COMMON_INSERT.xsl** file is displayed.

20. Connect the **inp1:Customers** source element to **db:X:APP_ID**.

21. Drag and drop the **populateXRefRow** function from the Component Palette to the connecting line.

22. Double-click the **populateXRefRow** icon.

    The Edit Function-populateXRefRow dialog is displayed.

23. Enter this information in the following fields:

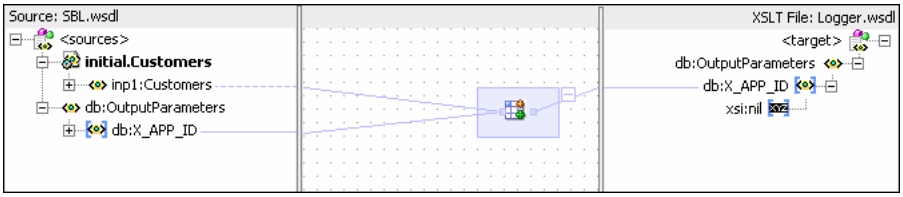
    - **xrefLocation**: `"customer.xref"`
    - **referenceColumnName**: `"Common"`
    - **referenceValue**: `$initial.Customers/inp1:Customers/inp1:Customer/inp1:Id`
    - **columnName**: `"SBL_78"`
    - **value**: `/db:OutputParameters/db:X_APP_ID`

- **mode**: `"LINK"`

24. Click **OK**.

    The **SBL_TO_COMMON_INSERT.xsl** file appears, as shown in Figure 46–36.

Figure 46–36 SBL_TO_COMMON_INSERT.xsl Transformation



25. From the **File** menu, select **Save All** and close the **SBL_TO_COMMON_INSERT.xsl** file.
26. In the **Synchronous Reply** section, click the **Assign Values** icon.

    The Assign Values dialog is displayed.

27. Click **Add**.

    The Assign Value dialog is displayed.

28. In the **From** section, select **Expression**.
29. Click the **Invoke Expression Builder** icon.

    The Expression Builder dialog is displayed.

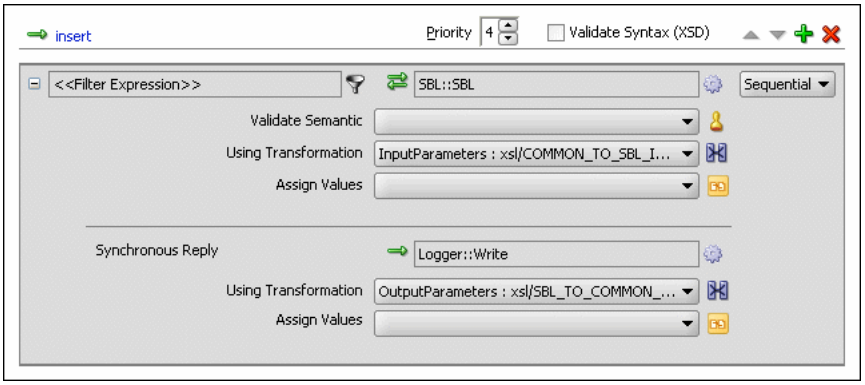
30. In the **Expression** field, enter the following expression and click **OK**.

```
concat('INSERT-',$in.OutputParameters/db:OutputParameters/db:X_APP_ID,'.xml')
```

31. In the **To** section, select **Property**.
32. Select the **jca.file.FileName** property and click **OK**.
33. Click **OK**.

    The **insert** operation section appears, as shown in Figure 46–37.

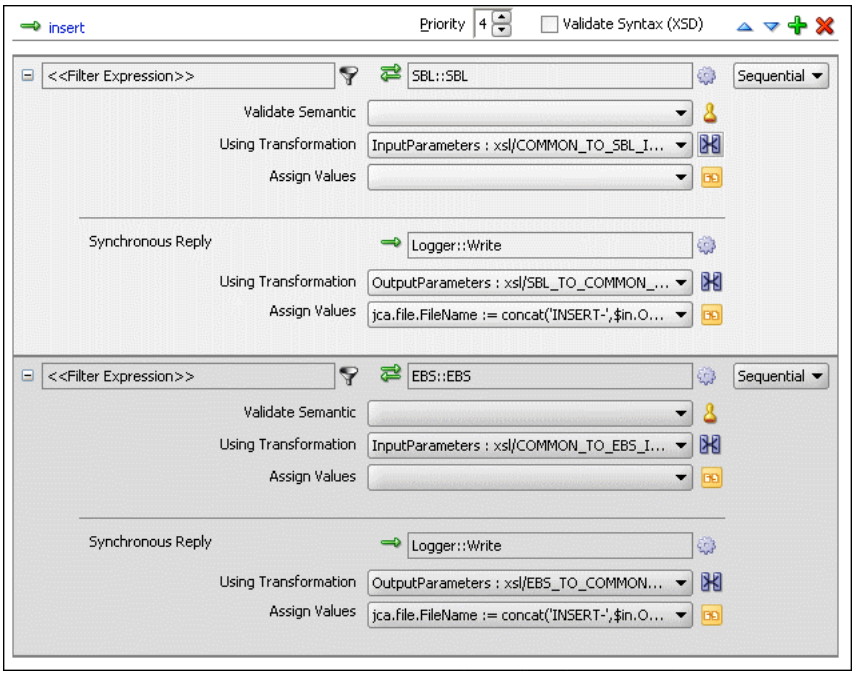
Figure 46–37 Insert Operation with SBL Target Service



34. From the **File** menu, select **Save All**.
35. Repeat Step 2 through Step 34 to specify another target service named **EBS** and its routing rules.

Figure 46–38 shows the **insert** operation section with **SBL** and **EBS** target services.

Figure 46–38 Insert Operation with SBL and EBS Target Services



To create routing rules for a delete operation:

Perform the following tasks to create the routing rules for a delete operation.

1. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.

2. Select **Service**.

   The Target Services dialog is displayed.

3. Navigate to **XrefCustApp** > **References** > **SBL**.

4. Select **SBL** and click **OK**.

5. Next to the **Transform Using** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.

6. Select **Create New Mapper File** and enter `COMMON_TO_SBL_DELETE.xsl`.

7. Click **OK**.

   A **COMMON_TO_SBL_DELETE.xsl** file is displayed.

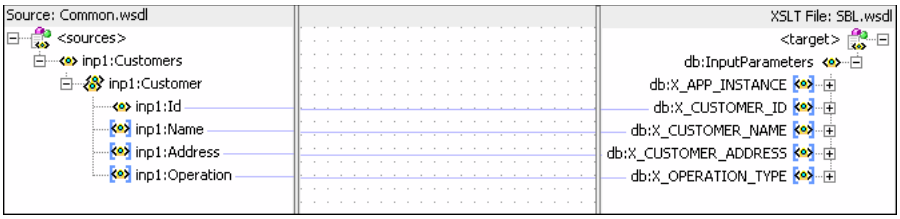
8. Drag and drop the **inp1:Customers** source element to the **db:InputParameters** target element.

   The Auto Map Preferences dialog is displayed.

9. Click **OK**.

   The transformation is created, as shown in Figure 46–39.

Figure 46–39 COMMON_TO_SBL_DELETE.xsl Transformation



10. Drag and drop the **lookupXRef** function from the Component Palette to the line connecting **inp1:id** and **db:XCUSTOMER_ID**.
11. Double-click the **lookupXRef** icon.

    The Edit Function: lookupXRef dialog is displayed.
12. Enter this information in the following fields:
    - **xrefLocation**: `"customer.xref"`
    - **referenceColumnName**: `"Common"`
    - **referenceValue**: `/inp1:Customers/inp1:Customer/inp1:Id`
    - **columnName**: `"SBL_78"`
    - **needException**: `false()`
13. Click **OK**.
14. From the **File** menu, select **Save All** and close the **COMMON_TO_SBL_DELETE.xsl** file.
15. In the **Synchronous Reply** section, click **Browse for target service operations**.

    The Target Type dialog is displayed.
16. Select **Service**.

    The Target Services dialog is displayed.
17. Navigate to **XrefCustApp** > **References** > **Logger**.
18. Select **Write** and click **OK**.
19. Next to the **Transform Using** field, click the **Transformation** icon.

    The Reply Transformation map dialog is displayed.
20. Select **Create New Mapper File** and enter `SBL_TO_COMMON_DELETE.xsl`.
21. Click **OK**.

    The **SBL_TO_COMMON_DELETE.xsl** file is displayed.
22. Connect the **db:X_APP_ID** source element to the **db:X:APP_ID** target.
23. Drag and drop the **markForDelete** function from the Component Palette to the connecting line.
24. Double-click the **markForDelete** icon.

    The Edit Function-markForDelete dialog is displayed.
25. Enter this information in the following fields:
    - **xrefLocation**: `"customer.xref"`
    - **columnName**: `"SBL_78"`

- **value**: `/db:OutputParameters/db:X_APP_ID`

26. Click **OK**.
27. From the **File** menu, select **Save All** and close the **SBL_TO_COMMON_DELETE.xsl** file.
28. In the **Synchronous Reply** section, click the **Assign Values** icon.

    The Assign Values dialog is displayed.
29. Click **Add**.

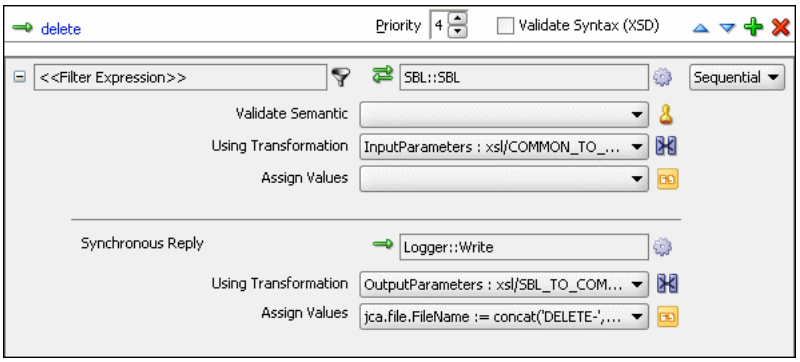
    The Assign Value dialog is displayed.
30. In the **From** section, select **Expression**.
31. Click the **Invoke Expression Builder** icon.

    The Expression Builder dialog is displayed.
32. In the **Expression** field, enter the following expression, and click **OK**.

    ```
    concat('DELETE-',$in.OutputParameters/db:OutputParameters/db:X_APP_ID,'.xml')
    ```
33. In the **To** section, select **Property**.
34. Select the **jca.file.FileName** property and click **OK**.
35. Click **OK**.

    The **delete** operation section appears, as shown in Figure 46–40.

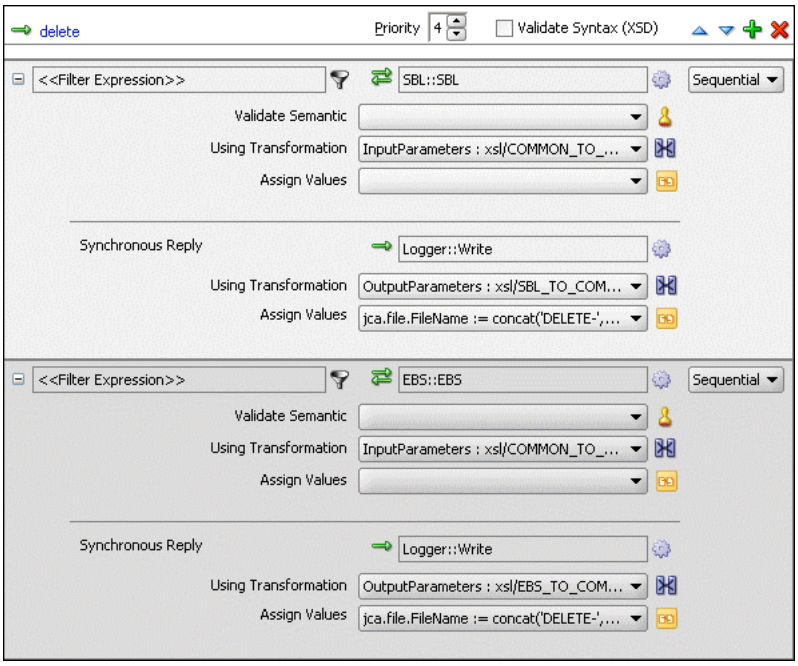
Figure 46–40 Delete Operation with SBL Target Service



36. From the **File** menu, select **Save All**.
37. Repeat Step 1 through Step 36 to specify another target service named **EBS** and specify the routing rules.

    Figure 46–41 shows the **delete** operation section with **SBL** and **EBS** target services.

Figure 46–41 Delete Operation with SBL and EBS Target Service



To create routing rules for the update operation:

Perform the following tasks to create routing rules for the update operation.

1. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.

2. Select **Service**.

   The Target Services dialog is displayed.

3. Navigate to **XrefCustApp**, **References** > **SBL**.

4. Select **SBL** and click **OK**.

5. Click the **Transformation** icon next to the **Transform Using** field.

   The Request Transformation map dialog is displayed.

6. Select **Create New Mapper File** and enter `COMMON_TO_SBL_UPDATE.xsl`.

7. Click **OK**.

   A **COMMON_TO_SBL_UPDATE.xsl** file is displayed.

8. Drag and drop the **inp1:Customers** source element to the **db:InputParameters** target element.

   The Auto Map Preferences dialog is displayed.

9. Click **OK**.

   The transformation is created, as shown in Figure 46–39.

10. Drag and drop the **lookupXRef** function from the Component Palette to the line connecting **inp1:id** and **db:XCUSTOMER_ID**.

11. Double-click the **lookupXRef** icon.

    The Edit Function: lookupXRef dialog is displayed.

12. Enter this information in the following fields:
    - **xrefLocation**: `"customer.xref"`
    - **referenceColumnName**: `"Common"`
    - **referenceValue**: `/inp1:Customers/inp1:Customer/inp1:Id`
    - **columnName**: `"SBL_78"`
    - **needException**: `true()`
13. Click **OK**.
14. From the **File** menu, select **Save All** and close the **COMMON_TO_SBL_UPDATE.xsl** file.
15. In the **Synchronous Reply** section, click **Browse for target service operations**.

    The Target Type dialog is displayed.
16. Select **Service**.

    The Target Services dialog is displayed.
17. Navigate to **XrefCustApp** > **References** > **Logger**.
18. Select **Write** and click **OK**.
19. Next to the **Transform Using** field, click the **Transformation** icon.

    The Reply Transformation map dialog is displayed.
20. Select **Create New Mapper File** and enter `SBL_TO_COMMON_UPDATE.xsl`.
21. Click **OK**.

    A **SBL_TO_COMMON_UPDATE.xsl** file is displayed.
22. Connect the **db:X:APP_ID** source element to **db:X:APP_ID**.
23. From the **File** menu, select **Save All** and close the **SBL_TO_COMMON_UPDATE.xsl** file.
24. In the **Synchronous Reply** section, click the **Assign Values** icon.

    The Assign Values dialog is displayed.
25. Click **Add**.

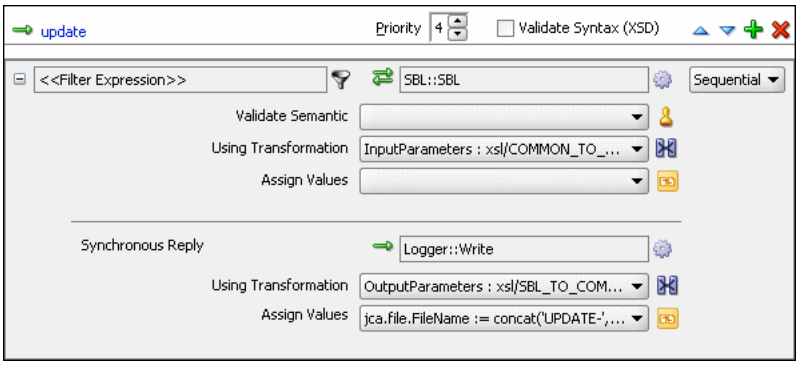
    The Assign Value dialog is displayed.
26. In the **From** section, select **Expression**.
27. Click the **Invoke Expression Builder** icon.

    The Expression Builder dialog is displayed.
28. In the **Expression** field, enter the following expression and click **OK**.

    ```
    concat('UPDATE-',$in.OutputParameters/db:OutputParameters/db:X_APP_ID,'.xml')
    ```
29. In the **To** section, select **Property**.
30. Select the **jca.file.FileName** property and click **OK**.
31. Click **OK**.

    The **update** operation section appears, as shown in Figure 46–42.

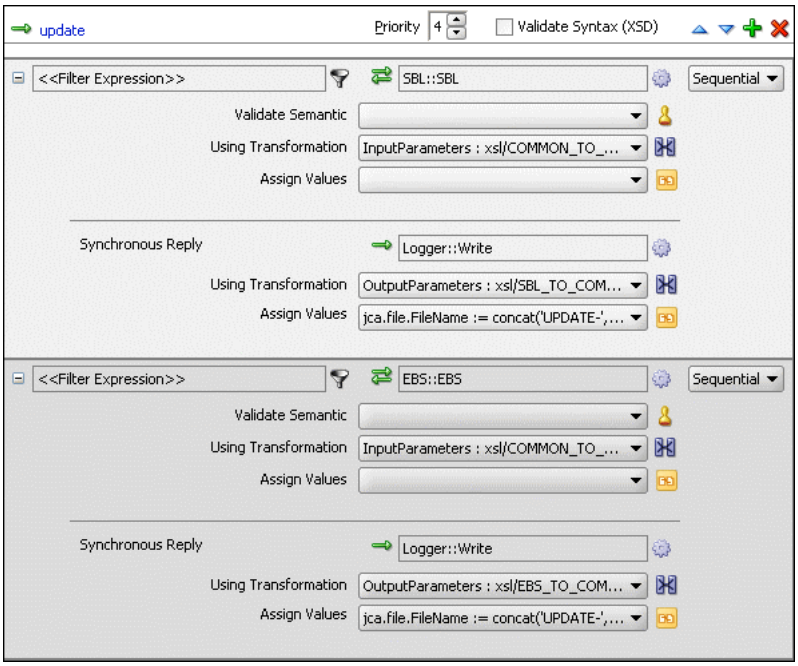
Figure 46–42 Update Operation with SBL Target Service



32. From the **File** menu, select **Save All**.
33. Repeat Step 1 through Step 32 to specify another target service named **EBS** and its routing rules.

    Figure 46–43 shows the **update** operation section with **SBL** and **EBS** target services.

Figure 46–43 Update Operation with SBL and EBS Target Service



To create routing rules for the UpdateID operation:

Perform the following tasks to create routing rules for the UpdateID operation.

1. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.
2. Select **Service**.

   The Target Services dialog is displayed.
3. Navigate to **XrefCustApp** > **References** > **SBL**.

4. Select **SBL** and click **OK**.
5. Next to the **Transform Using** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.
6. Select **Create New Mapper File** and enter `COMMON_TO_SBL_UPDATEID.xsl`.
7. Click **OK**.

   The **COMMON_TO_SBL_UPDATEID.xsl** file is displayed.
8. Drag and drop the **inp1:Customers** source element to the **db:InputParameters** target element.

   The Auto Map Preferences dialog is displayed.
9. Click **OK**.

   The transformation is created, as shown in Figure 46–39.
10. Drag and drop the **lookupXRef** function from the Component Palette to the line connecting **inp1:id** and **db:X_CUSTOMER_ID**.
11. Double-click the **lookupXRef** icon.

    The Edit Function: lookupXRef dialog is displayed.
12. Enter this information in the following fields:
    - **xrefLocation**: `customer.xref`
    - **referenceColumnName**: `Common`
    - **referenceValue**: `/inp1:Customers/inp1:Customer/inp1:Id`
    - **columnName**: `SBL_78`
    - **needException**: `false()`
13. Click **OK**.
14. From the **File** menu, select **Save All** and close the **COMMON_TO_SBL_UPDATEID.xsl** file.
15. In the **Synchronous Reply** section, click **Browse for target service operations**.

    The Target Type dialog is displayed.
16. Select **Service**.

    The Target Services dialog is displayed.
17. Navigate to **XrefCustApp** > **References** > **Logger**.
18. Select **Write** and click **OK**.
19. Next to the **Transform Using** field, click the **Transformation** icon.

    The Reply Transformation map dialog is displayed.
20. Select **Include Request in the Reply Payload**.
21. Click **OK**.

    The **SBL_TO_COMMON_UPDATEID.xsl** file is displayed.
22. Connect **inp1:Customers** source element to the **db:X:APP_ID**.
23. Drag and drop the **populateXRefRow** function from the Component Palette to the connecting line.

24. Double-click the **populateXRefRow** icon.

    The Edit Function-populateXRefRow dialog is displayed.

25. Enter this information in the following fields:

    - **xrefLocation**: `customer.xref`
    - **referenceColumnName**: `Common`
    - **referenceValue**: `$initial.Customers/inp1:Customers/inp1:Customer/inp1:Id`
    - **columnName**: `SBL_78`
    - **value**: `/db:OutputParameters/db:X_APP_ID`
    - **mode**: `UPDATE`

26. Click **OK**.

27. From the **File** menu, select **Save All** and close the **SBL_TO_COMMON_UPDATEID.xsl** file.

28. In the **Synchronous Reply** section, click the **Assign Values** icon.

    The Assign Values dialog is displayed.

29. Click **Add**.

    The Assign Value dialog is displayed.

30. In the **From** section, select **Expression**.

31. Click the **Invoke Expression Builder** icon.

    The Expression Builder dialog is displayed.

32. In the **Expression** field, enter the following expression and click **OK**.

    ```
    concat('UPDATEID-',$in.OutputParameters/db:OutputParameters/db:X_APP_ID,'.xml')
    ```

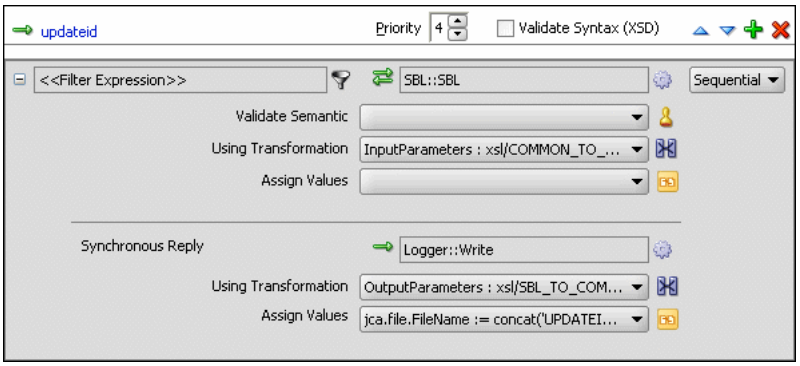
33. In the **To** section, select **Property**.

34. Select the **jca.file.FileName** property and click **OK**.

35. Click **OK**.

    The **updateid** operation section appears, as shown in Figure 46–44.

Figure 46–44 Updateid Operation with SBL Target Service

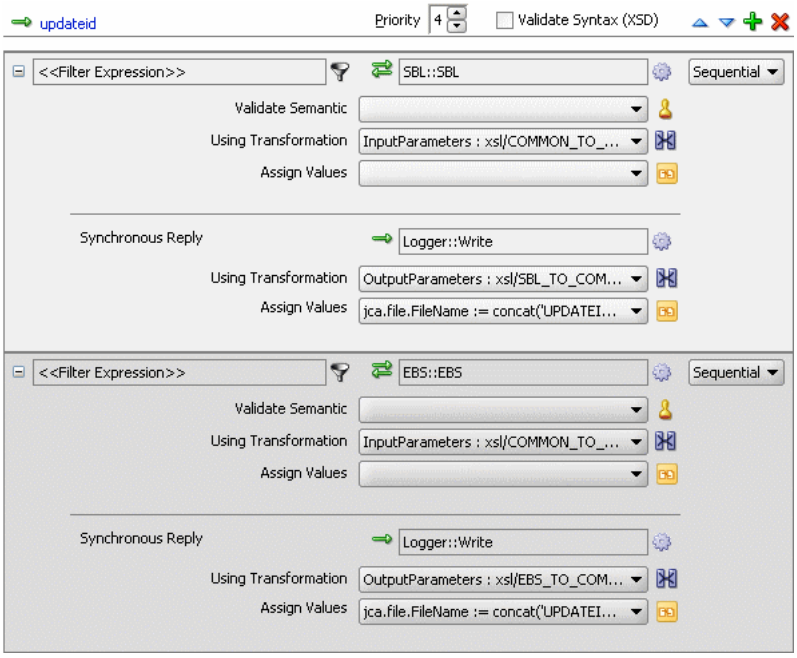


36. From the **File** menu, select **Save All**.

37. Repeat Step 1 through Step 36 to specify another target service named **EBS** and specify the routing rules.

    Figure 46–45 shows the **updateid** operation section with the **SBL** and **EBS** target services.

Figure 46–45 Updateid Operation with SBL and EBS Target Service



46.7.1.10 Task 10: 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."

46.7.1.11 Task 11: How to Deploy the Composite Application

Deploying the XrefCustApp composite application consists of the following steps:

- Creating an application deployment profile
- Deploying the application to the application server

For detailed information about these steps, see Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper."

46.7.2 How to Run and Monitor the XrefCustApp Application

After deploying the XrefCustApp application, you can run it by using any command from the `insert_sap_record.sql` file present in the `XrefCustApp/sql` folder. On successful completion, the records are inserted or updated in the `EBS` and `SBL` tables and the Logger reference writes the output to the `output.xml` file.

For monitoring the running instance, you can use the Oracle Enterprise Manager Fusion Middleware Control at the following URL:

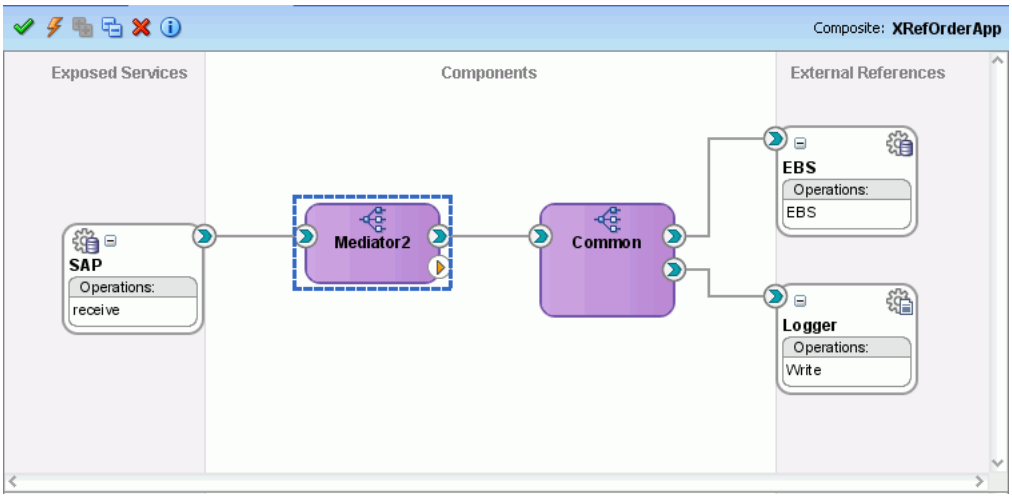
`http://hostname:port_number/em`

where *hostname* is the host on which you installed the Oracle SOA Suite infrastructure and *port_number* is the port running the service.

46.8 Creating and Running Cross Reference for 1M Functions

The cross reference use case implements an integration scenario between two end-system Oracle EBS and SAP instances. In this use case, the order passes from SAP to EBS. SAP represents the orders with a unique ID, whereas EBS splits the order into two orders: ID1 and ID2. This scenario is created using database adapters. When you poll the SAP table for updated or created records, an SAP instance is created. In EBS, the instance is simulated by a procedure and the table is populated. Figure 46–46 provides an overview of this use case.

Figure 46–46 XrefOrderApp Use Case in SOA Composite Editor



For downloading the sample files mentioned in this section, visit the following URL:

https://soasamples.samplecode.oracle.com/#mediator

46.8.1 How to Create the Use Case

This section provides the design-time tasks for creating, building, and deploying your SOA composite application. These tasks should be performed in the order in which they are presented.

46.8.1.1 Task 1: How to Configure the Oracle Database and Database Adapter

To configure the Oracle database and database adapter:

1. You need the `SCOTT` database account with password `TIGER` for this use case. You must ensure that the `SCOTT` account is unlocked.

   You can log in as `SYSDBA` and then run the `setup_user.sql` script available in the `XrefOrderApp1M/sql` folder to unlock the account.

2. Run the `create_schema.sql` script available in the `XrefOrderApp1M/sql` folder to create the tables required for this use case.

3. Run the `create_app_procedure.sql` script available in the `XrefOrderApp1M/sql` folder to create a procedure that simulates the various applications participating in this integration.

4. Run the `createschema_xref_oracle.sql` script available in the *`Oracle_Home`*`/rcu/integration/soainfra/sql/xref/` folder to create a cross reference table to store runtime cross reference data.
5. Copy the `ra.xml` and `weblogic-ra.xml` files from `$BEAHOME/META-INF` to the newly created directory called `META-INF` on your computer.
6. Edit the `weblogic-ra.xml` file, which is available in the `$BEAHOME/src/oracle/tip/adapter/db/test/deploy/weblogic/META-INF` folder for your SOA application, as follows:
   - Modify the property to `xADataSourceName` as follows:

     ```
     <property>
      <name>xADataSourceName</name>
      <value>jdbc/DBConnection1</value>
     </property>
     ```

   - Modify the `jndi-name` as follows:

     ```
     <jndi-name> eis/DB/DBConnection1</jndi-name>
     ```

   This sample uses `eis/DB/DBConnection1` to poll the `SAP` table for new messages and to connect to the procedure that simulates Oracle EBS and Siebel instances.
7. Package the `ra.xml` and `weblogic-ra.xml` files as a RAR file and deploy the RAR file by using Oracle WebLogic Server Administration Console.
8. Create a data source using the Oracle WebLogic Server Administration Console with the following values:
   - **jndi-name**=`jdbc/DBConnection1`
   - **user**=`scott`
   - **password**=`tiger`
   - **url**=`jdbc:oracle:thin:@host:port:service`
   - **connection-factory**
     factory-class=`oracle.jdbc.pool.OracleDataSource`
9. Create a data source using the Oracle WebLogic Server Administration Console with the following values:
   - **jndi-name**=`jdbc/xref`
   - **user**=`scott`
   - **password**=`tiger`
   - **url**=`jdbc:oracle:thin:@host:port:service`
   - **connection-factory**
     factory-class=`oracle.jdbc.pool.OracleDataSource`

46.8.1.2 Task 2: 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 `XRefOrderApp`, and then click **Next**.

   The Name your project page appears.
5. In the **Project Name** field, enter `XRefOrderApp` 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 updated with the new application and project and the SOA Composite Editor contains a blank project.
7. From the **File** menu, select **Save All**.

46.8.1.3 Task 3: How to Create a Cross Reference

After creating an application and a project for the use case, you must create a cross reference table.

To create a cross reference table:

1. In the Application Navigator, right-click the **XRefOrderApp** project and select **New**.
2. In the New Gallery dialog, expand the **SOA Tier** node, and then select the **Transformations** category.
3. In the **Items** list, select **Cross Reference(XREF)** and click **OK**.

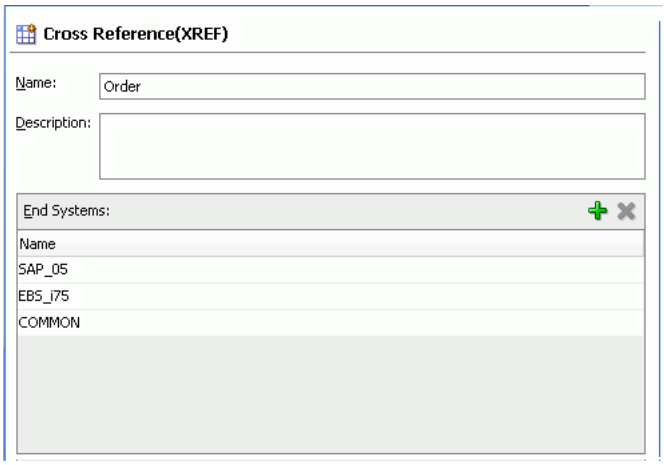
   The Create Cross Reference(XREF) File dialog is displayed.
4. In the **File Name** field, enter `order.xref`.
5. In the **End System** fields, enter `SAP_05` and `EBS_i75`.
6. Click **OK**.

   The Cross Reference Editor is displayed.
7. Click **Add**.

   A new row is added.
8. Enter **COMMON** as the **End System** name.

   The Cross Reference Editor appears, as shown in Figure 46–47.

Figure 46–47 Customer Cross Reference



9. From the **File** menu, select **Save All** and close the Cross Reference Editor.

46.8.1.4 Task 4: How to Create a Database Adapter Service

To create a database adapter service:

1. In the Component Palette, select **SOA**.
2. Select **Database Adapter** and drag it to the **Exposed Services** 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 `SAP`.
5. Click **Next**.

   The Service Connection page is displayed.
6. In the **Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter `eis/DB/DBConnection1`.
8. Click **Next**.

   The Operation Type page is displayed.
9. Select **Poll for New or Changed Records in a Table** and click **Next**.

   The Select Table page is displayed.
10. Click **Import Tables**.

    The Import Tables dialog is displayed.
11. Select **Scott** from the **Schema**.
12. In the **Name Filter** field, enter `%SAP%` and click **Query**.

    The **Available** field is populated with the **SAP_05** table name.
13. Double-click **SAP_05**.

    The **selected** field is populated with **SAP_05**.
14. Click **OK**.

The Select Table page now contains the **SAP_05** table.

15. Select **SAP_05** and click **Next**.

    The Define Primary Key page is displayed.

16. Select **ID** as the primary key and click **Next**.

    The Relationships page is displayed.

17. Click **Next**.

    The Attribute Filtering page is displayed.

18. Click **Next**.

    The After Read page is displayed.

19. Select **Update a Field in the [SAP_05] Table (Logical Delete)** and click **Next**.

    The Logical Delete page is displayed.

20. In the **Logical Delete** field, select **LOGICAL_DEL**.

21. In the **Read Value** field, enter `Y`.

22. In the **Unread Value** field, enter `N`.

    Figure 46–16 shows the Logical Delete page of the Adapter Configuration wizard.

23. Click **Next**.

    The Polling Options page is displayed.

24. Click **Next**.

    The Define Selection Criteria page is displayed.

25. Click **Next**.

    The Advanced Options page is displayed.

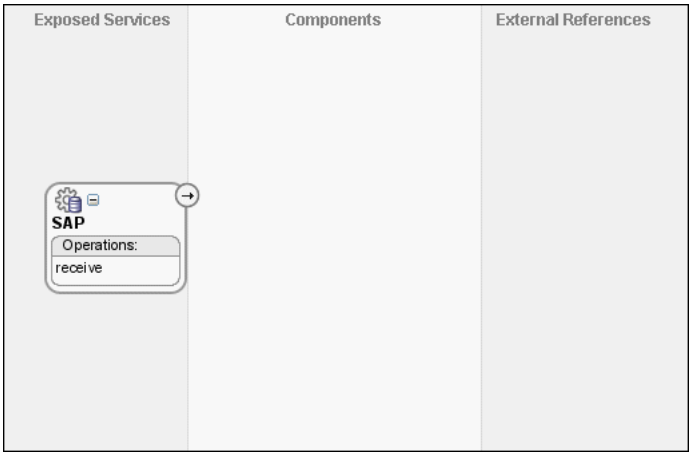
26. Click **Next**.

    The Finish page is displayed.

27. Click **Finish**.

    A database adapter service named **SAP** is created, as shown in Figure 46–48.

Figure 46–48 SAP Database Adapter Service in SOA Composite Editor



28. From the **File** menu, select **Save All**.

46.8.1.5 Task 5: How to Create an EBS External Reference

To create an EBS external reference:

1. In the Component Palette, select **SOA**.
2. Select **Database 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 `EBS`.
5. Click **Next**.

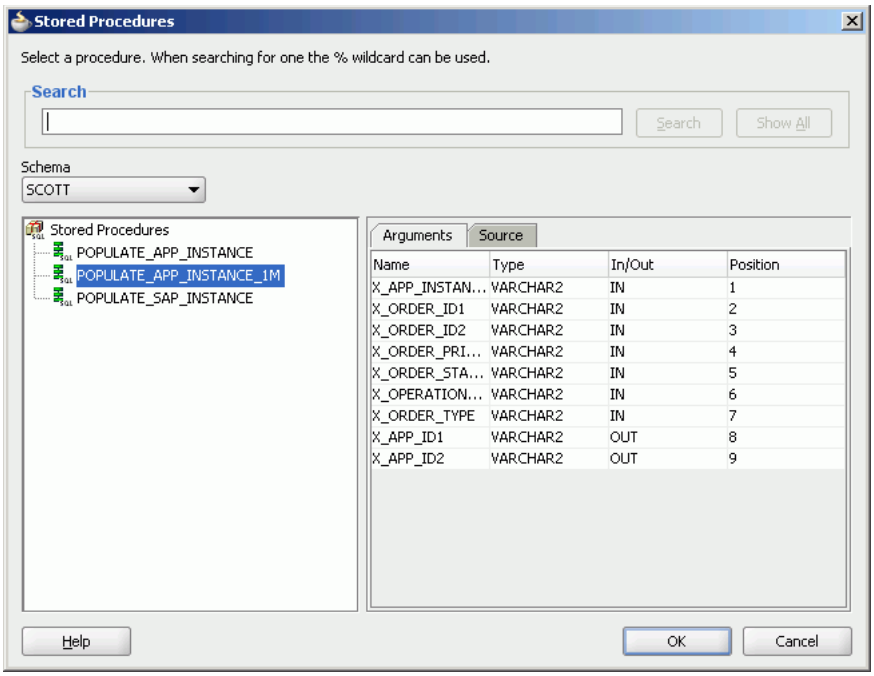
   The Service Connection page is displayed.
6. In the **Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter `eis/DB/DBConnection1`.
8. Click **Next**.

   The Operation Type page is displayed.
9. Select **Call a Stored Procedure or Function** and click **Next**.

   The Specify Stored Procedure page is displayed.
10. Select **Scott** from the **Schema**.
11. Click **Browse**.

    The Stored Procedures dialog is displayed.
12. Select **POPULATE_APP_INSTANCE_IM**, as shown in Figure 46–49.

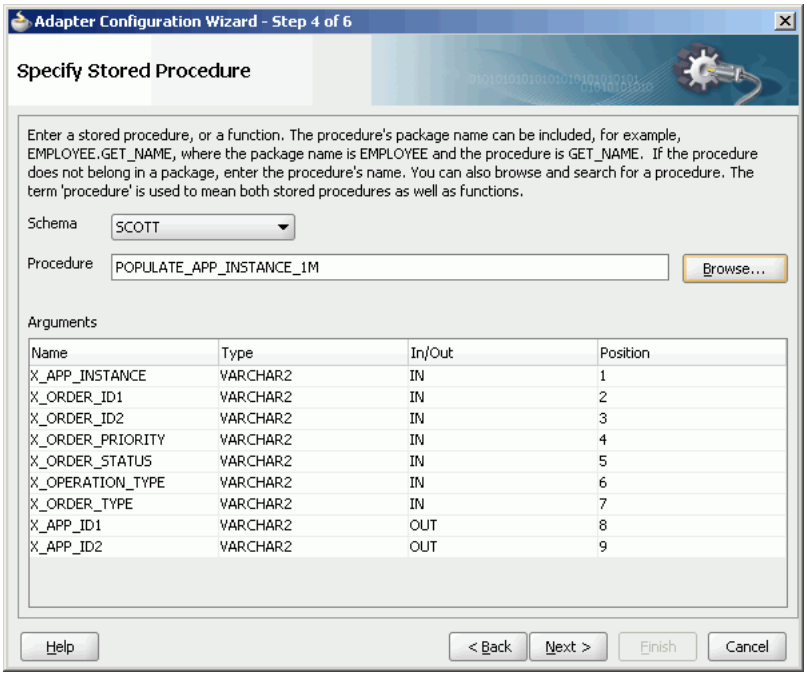
Figure 46–49 Stored Procedure Dialog



13. Click **OK**.

    The Specify Stored Procedure page appears, as shown in Figure 46–50.

Figure 46–50 Specify Stored Procedure Page of Adapter Configuration Wizard



14. Click **Next**.

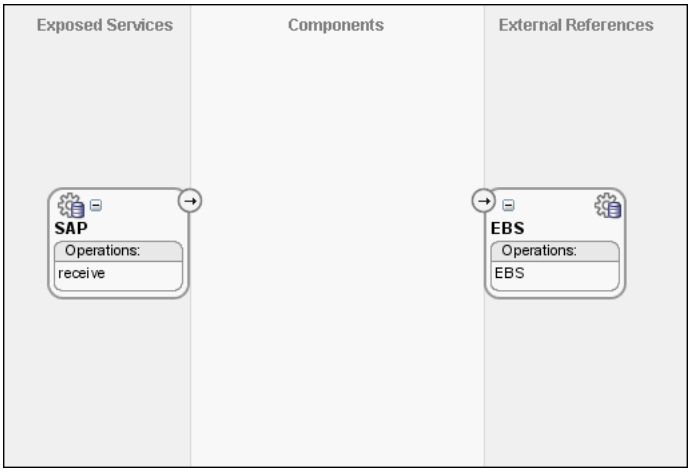
    The Advanced Options page is displayed.

15. Click **Next**.

16. The Finish page is displayed.
17. Click **Finish**.

    Figure 46–51 shows the **EBS** reference in the SOA Composite Editor.

Figure 46–51 EBS Reference in SOA Composite Editor



18. From the **File** menu, select **Save All**.

46.8.1.6 Task 6: How to Create a Logger File Adapter External Reference

To create a Logger file adapter external 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 `Logger`.
5. Click **Next**.

   The Adapter Interface page is displayed.
6. Click **Define from operation and schema (specified later)**.

   The Operation page is displayed.
7. In the **Operation Type** field, select **Write File**.
8. Click **Next**.

   The File Configuration page is displayed.
9. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory in which you want to write the files.
10. In the **File Naming Convention** field, enter `output.xml` and click **Next**.

    The Messages page is displayed.
11. Click **Search**.

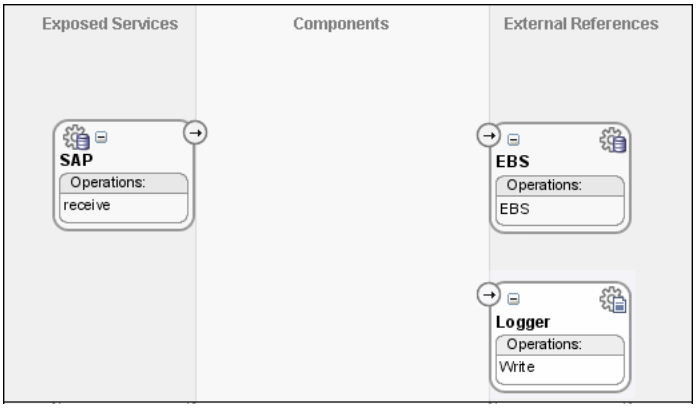
The Type Chooser dialog is displayed.

12. Navigate to **Type Explorer > Project Schema Files > SCOTT_POPULATE_APP_INSTANCE_1M.xsd**, and then select **OutputParameters**.
13. Click **OK**.
14. Click **Next.**

    The Finish page is displayed.
15. Click **Finish**.

    Figure 46–52 shows the **Logger** reference in the SOA Composite Editor.

Figure 46–52 Logger Reference in SOA Composite Editor



16. From the **File** menu. select **Save All**.

46.8.1.7 Task 7: How to Create an Oracle Mediator Service Component

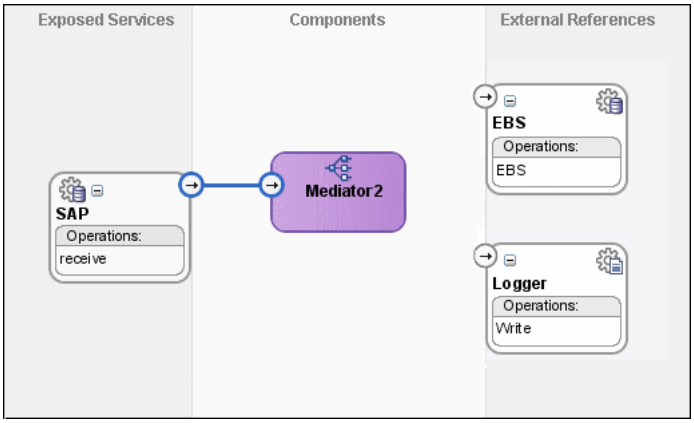
To create an Oracle Mediator service component:

1. Drag and drop a **Mediator** icon from the Component Palette to the **Components** swimlane.

   The Create Mediator dialog is displayed.
2. From the **Template** list, select **Define Interface Later**.
3. Click **OK**.

   An Oracle Mediator with name **Mediator2** is created.
4. Connect the **SAP** service to **Mediator2**, as shown in Figure 46–53.

Figure 46–53 SAP Service Connected to Mediator2



5. From the **File** menu. select **Save All**.
6. Drag and drop a **Mediator** icon from the Component Palette to the **Components** section of the SOA Composite Editor.

   The Create Mediator dialog is displayed.
7. From the **Template** list, select **Interface Definition From WSDL**.
8. Deselect **Create Composite Service with SOAP Bindings**.
9. To the right of the **WSDL File** field, click **Find Existing WSDLs**.
10. Navigate to and then select the **Common.wsdl** file. The **Common.wsdl** file is available in the **Samples** folder.
11. Click **OK**.
12. Click **OK**.

    An Oracle Mediator named **Common** is created.

46.8.1.8 Task 8: How to Specify Routing Rules for an Oracle Mediator Component

You must specify routing rules for following operations:

- Insert
- Update

To create routing rules for the insert operation:

1. Double-click the **Mediator2** Oracle Mediator.

   The Mediator Editor is displayed.
2. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.
3. Select **Service**.

   The Target Services dialog is displayed.
4. Navigate to **XrefOrderApp** > **Mediators** > **Common**, **Services** > **Common**.
5. Select **Insert** and click **OK**.
6. Click the **Filter** icon.

   The Expression Builder dialog is displayed.

7. In the **Expression** field, enter the following expression:

   ```
   $in.Sap05Collection/top:Sap05Collection/top:Sap05/top:operation='INSERT'
   ```

8. Click **OK**.
9. Next to the **Using Transformation** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.

10. Select **Create New Mapper File** and enter `SAP_TO_COMMON_INSERT.xsl`.
11. Click **OK**.

    An **SAP_TO_COMMON_INSERT.xsl** file is displayed.

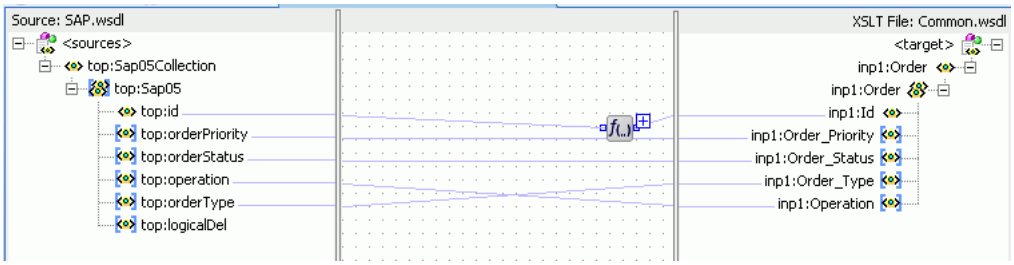
12. Drag and drop the **top:SAP05** source element to the **inp1:Order** target element.

    The Auto Map Preferences dialog is displayed.

13. From the **During Auto Map** options list, deselect **Match Elements Considering their Ancestor Names**.
14. Click **OK**.

    The transformation is created, as shown in Figure 46–54.

Figure 46–54 SAP_TO_COMMON_INSERT.xsl Transformation



15. From the Component Palette, select **Advanced**.
16. Select **XREF Functions**.
17. Drag and drop the **populateXRefRow1M** function from the Component Palette to the line connecting the **top:id** and **inp1:id** elements.
18. Double-click the **populateXRefRow1M** icon.

    The Edit Function-populateXRefRow dialog is displayed.

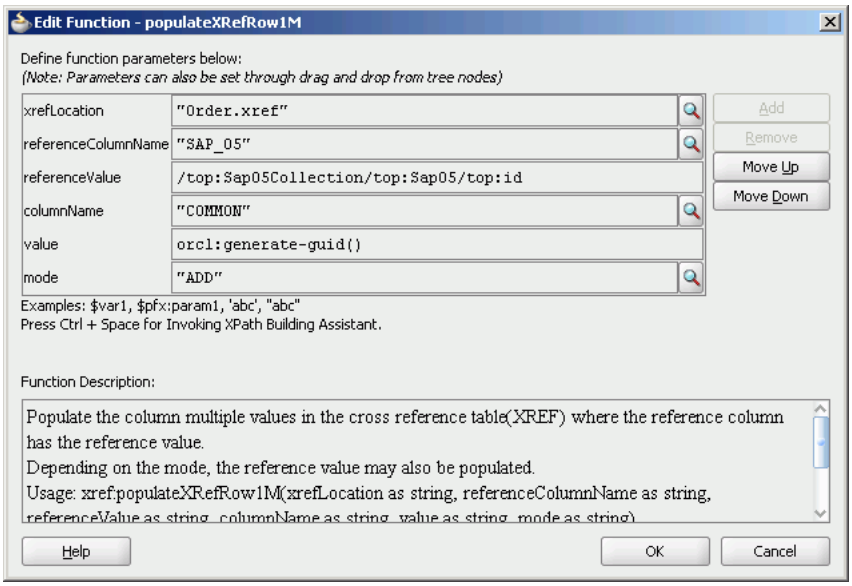
19. To the right of the **xrefLocation** field, click **Search**.

    The SOA Resource Lookup dialog is displayed.

20. Select **Order.xref** and click **OK**.
21. In the **referenceColumnName** field, enter `"SAP_05"` or click **Search** to select the column name.
22. In the **referenceValue** column, enter `/top:Sap05Collection/top:Sap05/top:id`.
23. In the **columnName** field, enter `"Common"` or click **Search** to select the column name.
24. In the **value** field, enter `orcl:generate-guid()`.
25. In the **mode** field, enter `"Add"` or click **Search** to select this mode.

Figure 46–55 shows the populated Edit Function – populateXRefRow1M dialog.

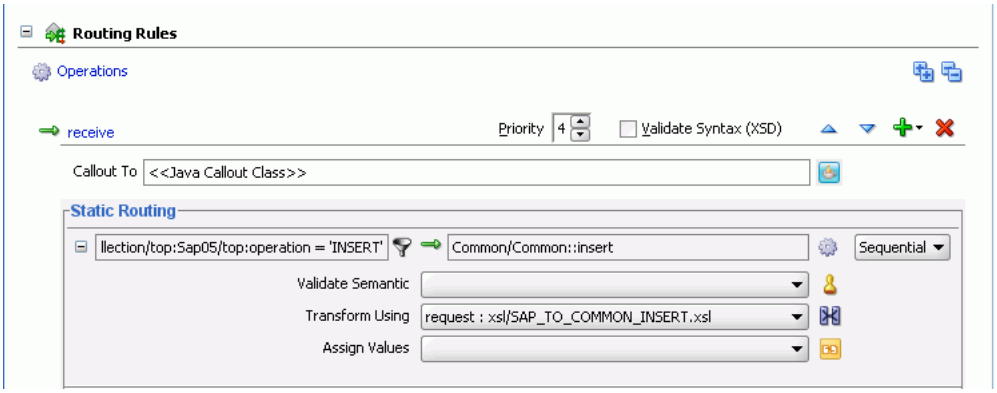
Figure 46–55 Edit Function – populateXRefRow1M Dialog: XrefOrderApp Use Case



26. Click **OK**.
27. From the **File** menu, select **Save All** and close the **SAP_TO_COMMON_INSERT.xsl** file.

    The **Routing Rules** section appears, as shown in Figure 46–56.

Figure 46–56 Routing Rules Section with Insert Operation



To create routing rules for the update operation:

Perform the following tasks to create routing rules for the update operation.

1. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.
2. Select **Service**.

   The Target Services dialog is displayed.
3. Navigate to **XrefOrderApp** > **Mediators** > **Common**, **Services** > **Common**.
4. Select **Update** and click **OK**.

5. Click the **Filter** icon.

   The Expression Builder dialog is displayed.

6. In the **Expression** field, enter the following expression:

   ```
   $in.Sap05Collection/top:Sap05Collection/top:Sap05/top:operation='UPDATE'
   ```

7. Click **OK**.

8. Next to the **Transform Using** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.

9. Select **Create New Mapper File** and enter `SAP_TO_COMMON_UPDATE.xsl`.

10. Click **OK**.

    An **SAP_TO_COMMON_UPDATE.xsl** file is displayed.

11. Drag and drop the **top:Sap05** source element to the **inp1:Order** target element.

    The Auto Map Preferences dialog is displayed.

12. Click **OK**.

13. From the Component Palette, select **Advanced**.

14. Select **XREF Functions**.

15. Drag and drop the **lookupXRef** function from the Component Palette to the line connecting the **top:id** and **inp1:id** elements.

16. Double-click the **lookupXRef** icon.

    The Edit Function-lookupXRef dialog is displayed.

17. To the right of the **xrefLocation** field, click **Search**.

    The SOA Resource Lookup dialog is displayed.

18. Select **customer.xref** and click **OK**.

19. In the **referenceColumnName** field, enter `"SAP_05"` or click **Search** to select the column name.

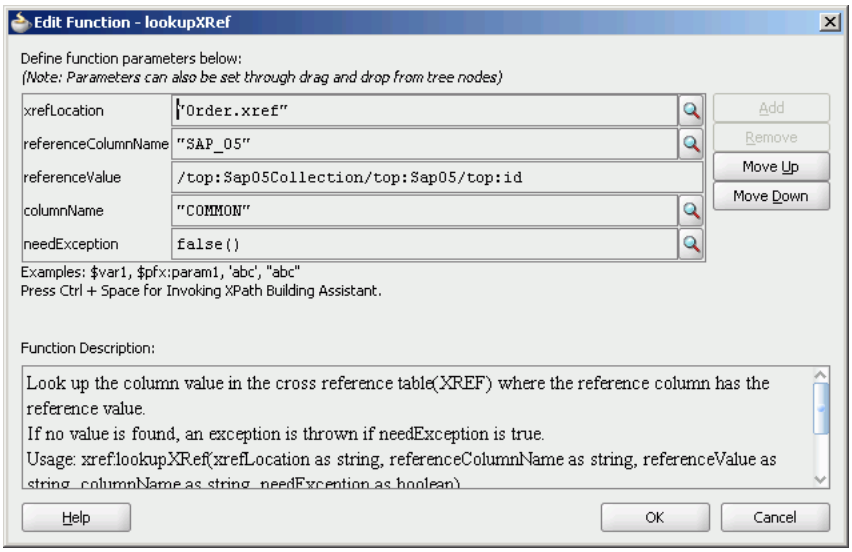
20. In the **referenceValue** column, enter `/top:Sap05Collection/top:Sap05/top:id`.

21. In the **columnName** field, enter `"COMMON"` or click **Search** to select the column name.

22. In the **needException** field, enter `true()` or click **Search** to select this mode.

    Figure 46–57 shows the populated Edit Function – looupXRef dialog.

Figure 46–57 Edit Function – looupXRef Dialog: XRefOrderApp Use Case

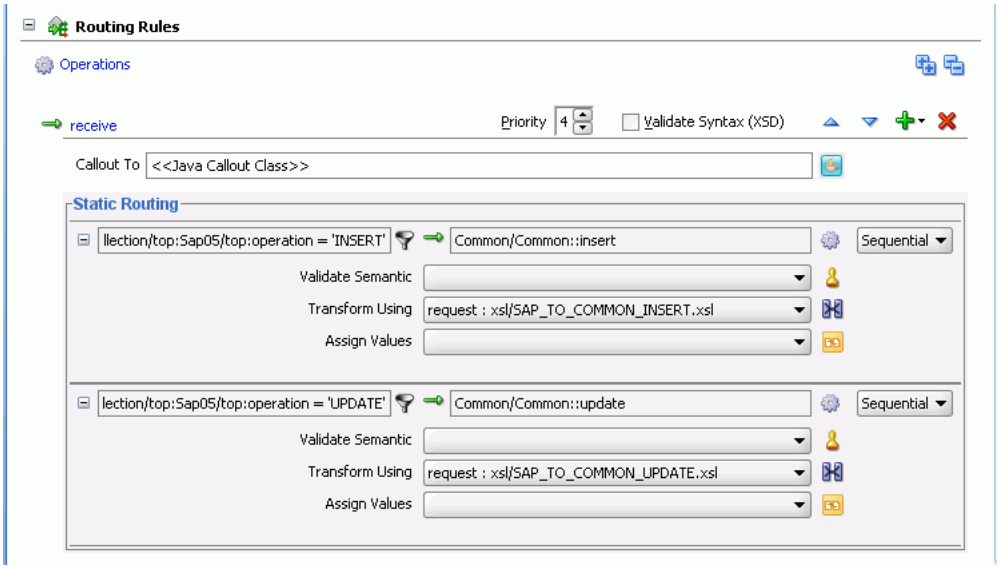


23. Click **OK**.

24. From the **File** menu, select **Save All** and close the **SAP_TO_COMMON_UPDATE.xsl** file.

    The **Routing Rules** section appears, as shown in Figure 46–58.

Figure 46–58 Insert Operation and Update Operation



46.8.1.9 Task 9: How to Specify Routing Rules for the Common Oracle Mediator

You must specify routing rules for the following operations of the Common Oracle Mediator:

- Insert
- Update

To create routing rules for the insert operation:

1. Double-click the **Common** Oracle Mediator.

   The Mediator Editor is displayed.

2. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.

3. Select **Service**.

   The Target Services dialog is displayed.

4. Navigate to **XrefOrderApp** > **References** > **EBS**.

5. Select **EBS** and click **OK**.

6. Next to the **Transform Using** field, click the **Transformation** icon.

   The Request Transformation map dialog is displayed.

7. Select **Create New Mapper File** and enter `COMMON_TO_EBS_INSERT.xsl`.

8. Click **OK**.

   A **COMMON_TO_EBS_INSERT.xsl** file is displayed.

9. Drag and drop the **inp1:Order** source element to the **db:InputParameters** target element.

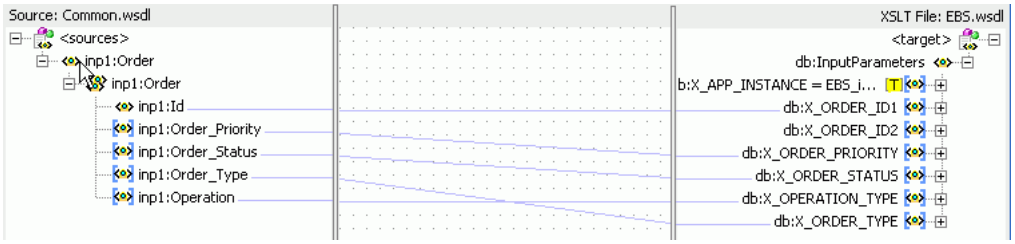
   The Auto Map Preferences dialog is displayed.

10. Set the value of the **db:X_APP_INSTANCE** node on the right side to **EBS_i75**.

    Click **OK**.

    The transformation is created, as shown in Figure 46–59.

Figure 46–59 COMMON_TO_EBS_INSERT.xsl Transformation



11. From the **File** menu, select **Save All** and close the **COMMON_TO_EBS_INSERT.xsl** file.

12. In the **Synchronous Reply** section, click **Browse for target service operations**.

    The Target Type dialog is displayed.

13. Select **Service**.

    The Target Services dialog is displayed.

14. Navigate to **XrefOrderApp** > **References** > **Logger**.

15. Select **Write** and click **OK**.

16. Next to the **Transform Using** field, click the **Transformation** icon.

    The Reply Transformation map dialog is displayed.

17. Select **Create New Mapper File** and enter `EBS_TO_COMMON_INSERT.xsl`.

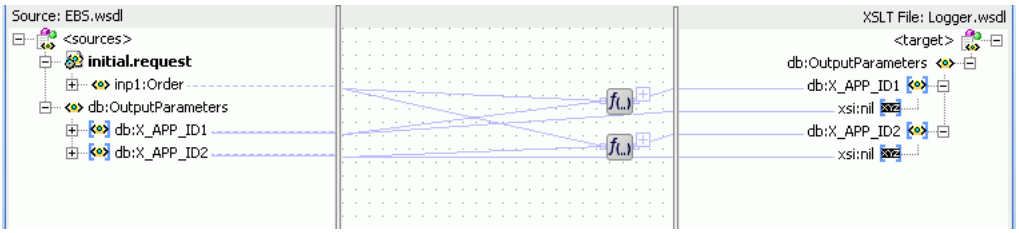
18. Select **Include Request in the Reply Payload**.
19. Click **OK**.

    An **EBS_TO_COMMON_INSERT.xsl** file is displayed.
20. Connect the **inp1:Order** source element to **db:X:APP_ID**.
21. Drag and drop the **populateXRefRow** function from the Component Palette to the connecting line.
22. Double-click the **populateXRefRow** icon.

    The Edit Function-populateXRefRow dialog is displayed.
23. Enter this information in the following fields:
    - **xrefLocation**: `order.xref`
    - **referenceColumnName**: `Common`
    - **referenceValue**: `$initial.Customers/inp1:Customers/inp1:Order/inp1:Id`
    - **columnName**: `EBS_75`
    - **value**: `/db:OutputParameters/db:X_APP_ID`
    - **mode**: `LINK`
24. Click **OK**.

    The **EBS_TO_COMMON_INSERT.xsl** file appears, as shown in Figure 46–60.

Figure 46–60 EBS_TO_COMMON_INSERT.xsl Transformation



25. From the **File** menu, select **Save All** and close the **EBS_TO_COMMON_INSERT.xsl** file.
26. In the **Synchronous Reply** section, click the **Assign Values** icon.

    The Assign Values dialog is displayed.
27. Click **Add**.

    The Assign Value dialog is displayed.
28. In the **From** section, select **Expression**.
29. Click the **Invoke Expression Builder** icon.

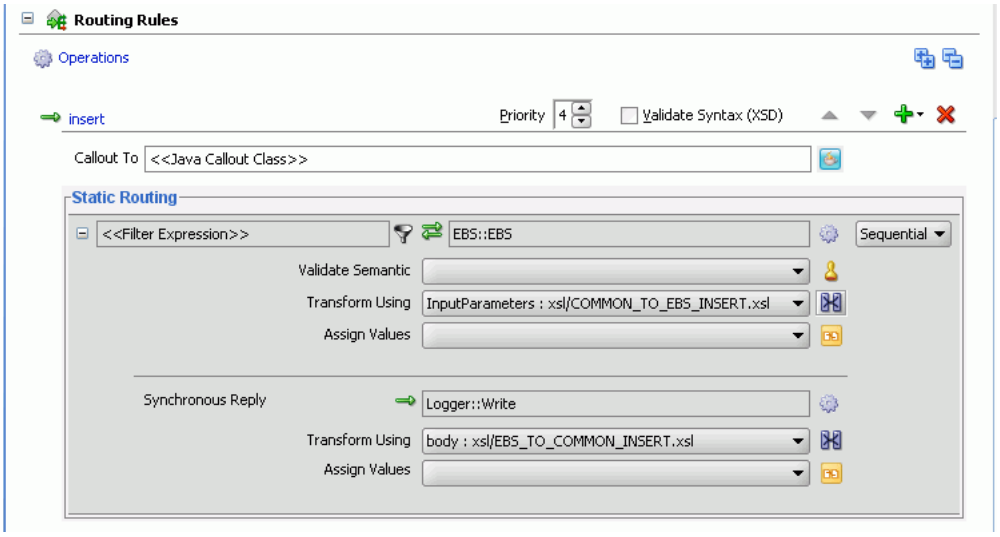
    The Expression Builder dialog is displayed.
30. In the **Expression** field, enter the following expression and click **OK**.

    ```
    concat('INSERT-',$in.OutputParameters/db:OutputParameters/db:X_APP_ID,'.xml')
    ```
31. In the **To** section, select **Property**.
32. Select the **jca.file.FileName** property and click **OK**.

33. Click **OK**.

    The **insert** operation section appears, as shown in Figure 46–61.

Figure 46–61 Insert Operation with EBS Target Service



34. From the **File** menu, select **Save All**.

To create routing rules for the update operation:

Perform the following tasks to create routing rules for the update operation.

1. In the **Routing Rules** section, click the **Create a new Routing Rule** icon.

   The Target Type dialog is displayed.

2. Select **Service**.

   The Target Services dialog is displayed.

3. Navigate to **XrefOrderApp** > **References** > **EBS**.

4. Select **EBS** and click **OK**.

5. Click the **Transformation** icon next to the **Transform Using** field.

   The Request Transformation map dialog is displayed.

6. Select **Create New Mapper File** and enter `COMMON_TO_EBS_UPDATE.xsl`.

7. Click **OK**.

   The **COMMON_TO_EBS_UPDATE.xsl** file is displayed.

8. Drag and drop the **inp1:Orders** source element to the **db:InputParameters** target element.

   The Auto Map Preferences dialog is displayed.

9. Click **OK**.

   The transformation is created, as shown in Figure 46–39.

10. Drag and drop the **lookupXRef** function from the Component Palette to the line connecting **inp1:id** and **db:X_APP_ID**.

11. Double-click the **lookupXRef** icon.

The Edit Function: lookupXRef dialog is displayed.

12. Enter this information in the following fields:
    - **xrefLocation**: `order.xref`
    - **referenceColumnName**: `Common`
    - **referenceValue**: `/inp1:Customers/inp1:Order/inp1:Id`
    - **columnName**: `EBS_i75`
    - **needException**: `true()`
13. Click **OK**.
14. From the **File** menu, select **Save All** and close the **COMMON_TO_EBS_UPDATE.xsl** file.
15. In the **Synchronous Reply** section, click **Browse for target service operations**.

    The Target Type dialog is displayed.
16. Select **Service**.

    The Target Services dialog is displayed.
17. Navigate to **XrefOrderApp** > **References** > **Logger**.
18. Select **Write** and click **OK**.
19. Next to the **Transform Using** field, click the **Transformation** icon.

    The Reply Transformation map dialog is displayed.
20. Select **Create New Mapper File** and enter `EBS_TO_COMMON_UPDATE.xsl`.
21. Click **OK**.

    The **EBS_TO_COMMON_UPDATE.xsl** file is displayed.
22. Connect the **db:X:APP_ID** source element to **db:X:APP_ID**.
23. From the **File** menu, select **Save All** and close the **EBS_TO_COMMON_UPDATE.xsl** file.
24. In the **Synchronous Reply** section, click the **Assign Values** icon.

    The Assign Values dialog is displayed.
25. Click **Add**.

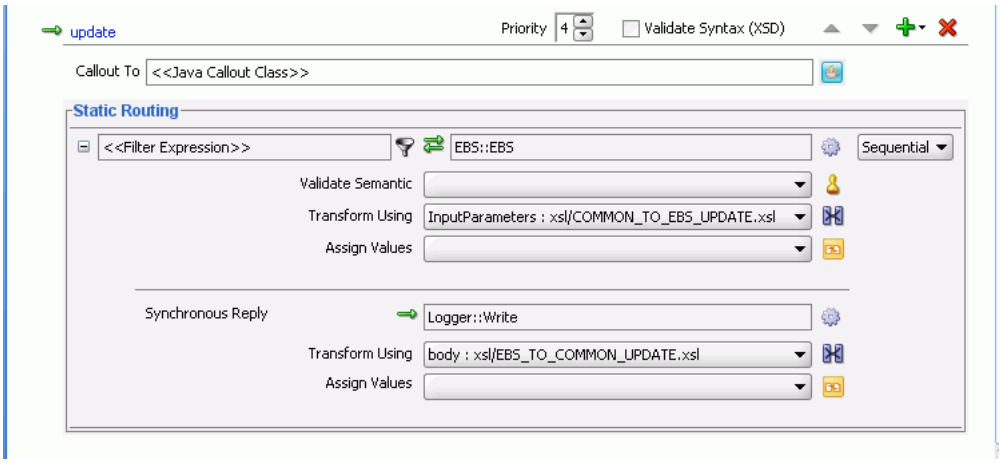
    The Assign Value dialog is displayed.
26. In the **From** section, select **Expression**.
27. Click the **Invoke Expression Builder** icon.

    The Expression Builder dialog is displayed.
28. In the **Expression** field, enter the following expression, and click **OK**.

    ```
    concat('UPDATE-',$in.OutputParameters/db:OutputParameters/db:X_APP_ID,'.xml')
    ```
29. In the **To** section, select **Property**.
30. Select the **jca.file.FileName** property and click **OK**.
31. Click **OK**.

    The **update** operation section appears, as shown in Figure 46–62.

Figure 46–62 Update Operation with EBS Target Service



32. From the **File** menu, select **Save All**.

46.8.1.10 Task 10: How to Configure 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."

46.8.1.11 Task 11: How to Deploy the Composite Application

Deploying the **XrefOrderApp** composite application to the application server consists of the following steps:

- Creating an application deployment profile
- Deploying the application to the application server

For detailed information about these steps, see Section 40.7.1, "Deploying a Single SOA Composite in Oracle JDeveloper."

47 Defining Composite Sensors

This chapter describes how to define composite sensors in a SOA composite application.

This chapter includes the following sections:

47.1 Introduction to Composite Sensors

Composite sensors provide a method for implementing trackable fields on messages. Composite sensors enable you to perform the following tasks:

- Monitor incoming and outgoing messages.
- Specify composite sensor details in the search utility of the Instances page of a SOA composite application in Oracle Enterprise Manager Fusion Middleware Control. This action enables you to locate a particular instance.
- Publish JMS data computed from incoming and outgoing messages.

You define composite sensors on service and reference binding components in Oracle JDeveloper. This functionality is similar to variable sensors in BPEL processes. During runtime, composite sensor data is persisted in the database.

47.1.1 Restrictions on Use of Composite Sensors

Note the following restrictions on the use of composite sensors:

- Functions can only be used with the payload. For example, XPath functions such as `concat()` and others cannot be used with properties.
- Any composite sensor that uses expressions always captures values as strings. This action makes the search possible only with the `like` comparison operator. Also, even if the value is a number, you cannot use other logical operators such as <, >, =, and any combination of these.
- Composite sensors only support two types of sensor actions: Enterprise Manager and JMS.
- Header-based sensors are only supported for web service bindings.
- Sensor actions for Oracle B2B, service data objects (SDOs), web services invocation framework (WSIF), and Oracle Business Activity Monitoring bindings are not supported.

- Sensor values can only be one of the following types.
  1. The following scalar types:
     - `STRING`
     - `NUMBER`
     - `DATE`
     - `DATE_TIME`
  2. Complex XML elements
- When creating an XPath expression for filtering, all functions that return a node set must be explicitly cast as a string:

```
xpath20:upper-case(string($in.request/inp1:updateOrderStatus/inp1:orderStatus)
) = "PENDING"
```

47.2 Adding Composite Sensors

You add sensors to service or reference binding components of a SOA composite application in the SOA Composite Editor.

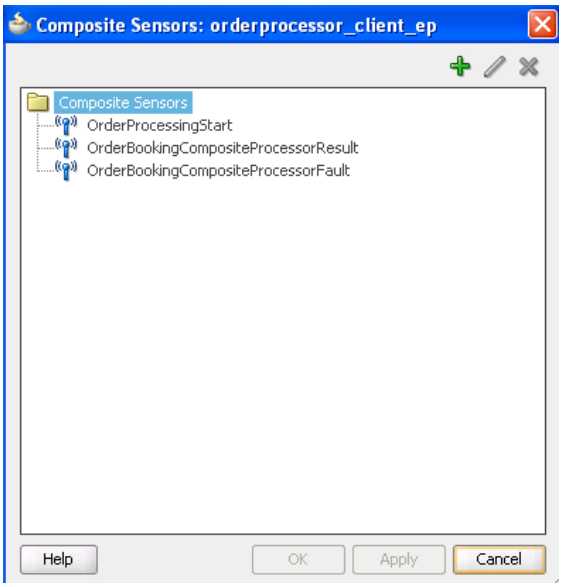
47.2.1 How to Add Composite Sensors

To add composite sensors:

1. Use one of the following options to add a composite sensor in the SOA Composite Editor.
   a. Right-click a specific service or reference binding component in the **Exposed Services** or **External References** swimlane to which to add a composite sensor, and select **Composite Sensors**.

      The Composite Sensors dialog for the selected binding component appears, as shown in Figure 47–1.

Figure 47–1 Composite Sensors Dialog for the Selected Binding Component



b. Click the **Add** icon.

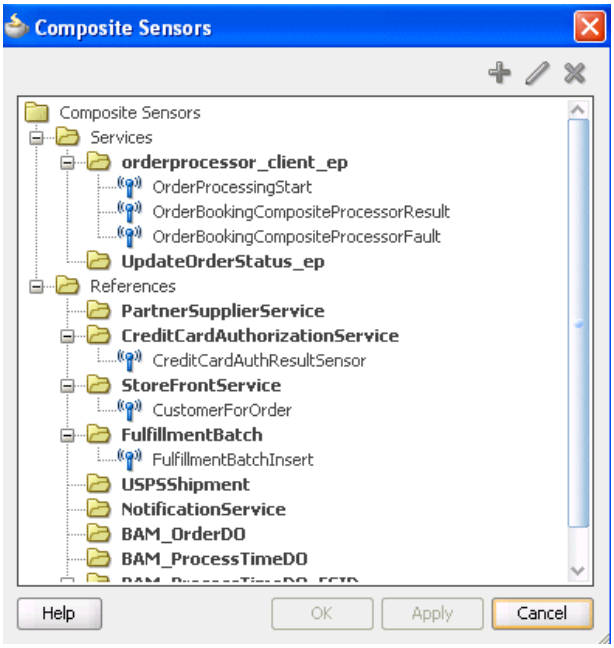
or

a. Click the **Composite Sensor** icon above the SOA Composite Editor, as shown in Figure 47–2.

Figure 47–2 Composite Sensor Icon

The Composite Sensors dialog for the SOA composite application appears, as shown in Figure 47–3.

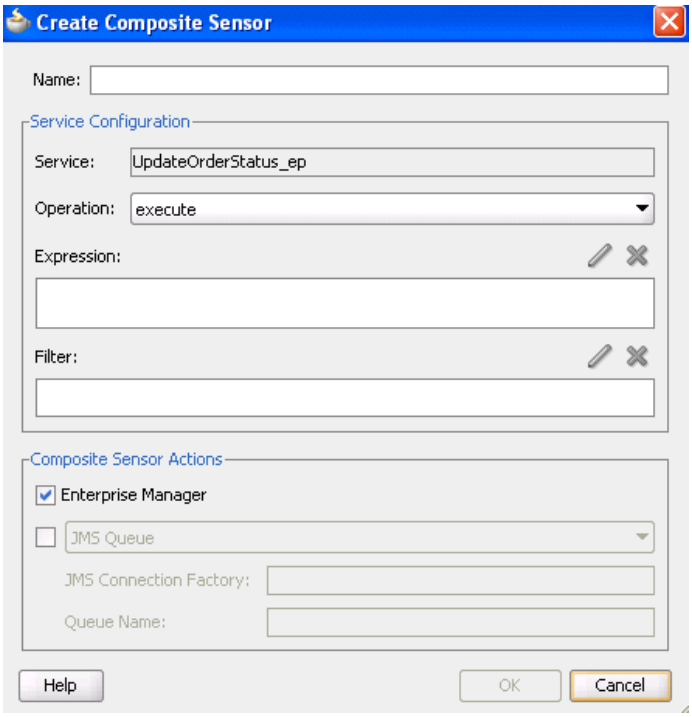
Figure 47–3 Composite Sensors Dialog



b. Select the specific service or reference to which to add a composite sensor, then click the **Add** icon.

The Create Composite Sensor dialog appears, as shown in Figure 47–4.

Figure 47–4 Create Composite Sensor Dialog



2. Enter the details shown in Table 47–1.

Table 47–1 Create Composite Sensor Dialog

Name	Description
Name	Enter a name for the composite sensor. You must enter a name to enable the **Edit** icon of the **Expression** field.
Service	Displays the name of the service. This field only displays if you are creating a composite sensor for a service binding component. This field cannot be edited. Service sensors monitor the messages that the service receives from the external world or from another composite application.
Reference	Displays the name of the reference. This field only displays if you are creating a composite sensor for a reference binding component. This field cannot be edited. Reference sensors monitor the messages that the reference sends to the external world or to another composite application.
Operation	Select the operation for the port type of the service or reference.

Table 47–1 (Cont.) Create Composite Sensor Dialog

Name	Description
Expression	Click the Edit icon to invoke a dropdown list for selecting the type of expression to create: ■ **Variables**: Select to create an expression value for a variable. See Section 47.2.2, "How to Add a Variable" for instructions. ■ **Expression**: Select to invoke the Expression Builder dialog for creating an XPath expression. See Section 47.2.3, "How to Add an Expression" for instructions. ■ **Properties**: Select to create an expression value for a normalized message header property. These are the same properties that display under the **Properties** tab of the invoke activity, receive activity, reply activity, OnEvent branch of a scope activity (in BPEL 2.0), and OnMessage branch of a pick activity and a scope activity (in BPEL 2.0). See Section 47.2.4, "How to Add a Property" for instructions.
Filter	Click the **Edit** icon to invoke the Expression Builder dialog to create an XPath filter for the expression. You must first create an expression to enable this field. For example, you may create an expression for tracking purchase order amounts over `10,000`: `$in.inDict/tns:inDict/ns2:KeyValueOfstringstring/ns2:Value > 10000.00`
Composite Sensor Actions	Displays the supported sensor actions. This feature enables you to store runtime sensor data. You can select both **Enterprise Manager** and either **JMS Queue** or **JMS Topic**. ■ **Enterprise Manager** Select to make runtime sensor data searchable in the **Instances** tab of a SOA composite application in Oracle Enterprise Manager Fusion Middleware Control. This selection is the same as the **DBSensorAction** selection of previous releases. **Note**: When **Enterprise Manager** is selected, sensor data is sent to the trackable fields tables. When it is not selected, data is not sent. However, in both cases, Oracle Enterprise Manager Fusion Middleware Control still displays the fields that enable you to search for composite instances based on that sensor. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*. ■ **JMS Queue** Select to store composite sensor data (XML payload) in a JMS queue. You must specify the JMS connection factory and queue name. ■ **JMS Topic** Select to store composite sensor data (XML payload) in a JMS topic. You must specify the JMS connection factory and topic name. **Notes**: The **JMS Queue** and **JMS Topic** selections enable the composite sensor data (XML payload) to be used by other consumers, including Oracle Business Activity Monitoring and Oracle Complex Event Processing (CEP). Both selections use the native JMS support provided with Oracle WebLogic Server, and *not* the Oracle SOA Suite JMS adapter described in *Oracle Fusion Middleware User's Guide for Technology Adapters*. You can view JMS messages in the Oracle WebLogic Server Administration Console. For information about using sensor data with Oracle Business Activity Monitoring, see Chapter 50, "Integrating Oracle BAM with SOA Composite Applications."

3. Click **Apply** to add more composite sensors.
4. Click **OK** when complete.

   A **composite sensor** icon displays on the service or reference binding component, as shown in Figure 47–5.

Figure 47–5 Sensor Icon

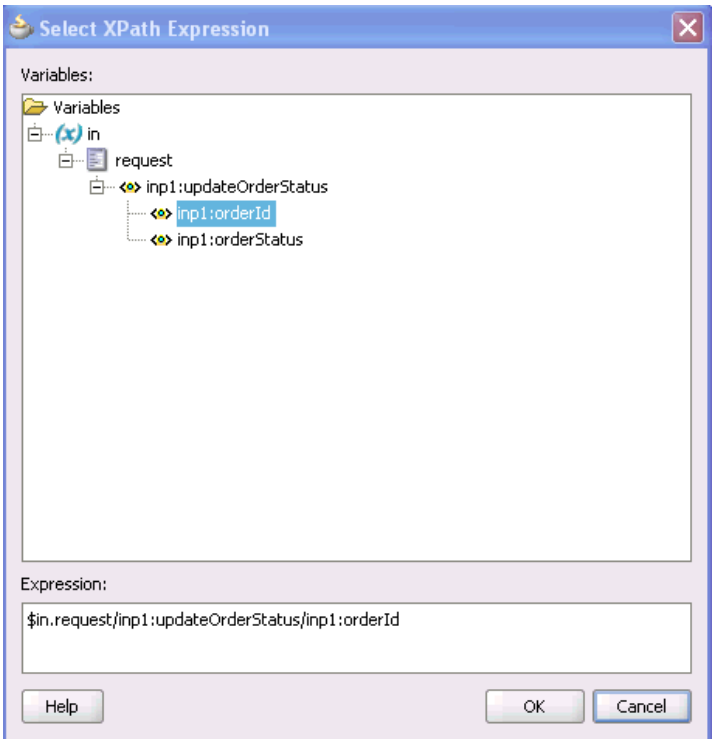
47.2.2 How to Add a Variable

The Select XPath Expression dialog shown in Figure 47–6 enables you to select an element for tracking.

To add a variable:

1. Expand the tree and select the element to track.

Figure 47–6 Variables



2. Click **OK** when complete.

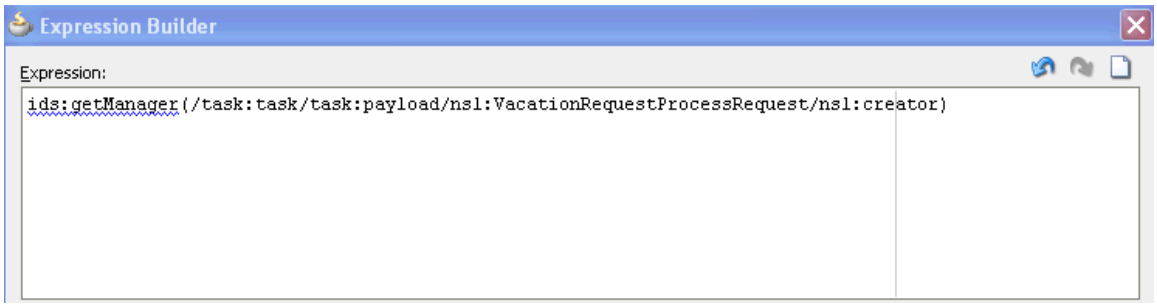
47.2.3 How to Add an Expression

The Select Properties shown in Figure 47–7 enables you to create an expression for tracking.

To add an expression:

1. Build an XPath expression of an element to track.

Figure 47–7 Expression



2. Click **OK** when complete.

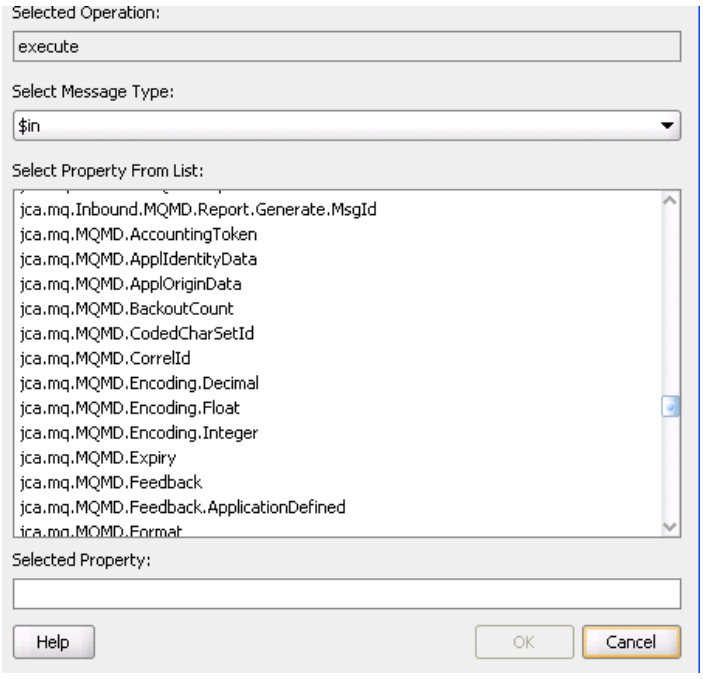
47.2.4 How to Add a Property

The Select Property shown in Figure 47–8 enables you to select a normalized message header property for tracking.

To add a property:

1. Select a normalized message header property to track.

Figure 47–8 Properties



2. Click **OK** when complete.

47.3 Monitoring Composite Sensor Data During Runtime

During runtime, composite sensor data can be monitored in Oracle Enterprise Manager Fusion Middleware Control:

- Composite sensor data displays in the flow trace of a SOA composite application.
- Composite sensor data can be searched for in the Instances page of a SOA composite application.

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

48

Using Two-Layer Business Process Management (BPM)

This chapter describes how to use two-layer Business Process Management (BPM). Two-layer BPM enables you to create dynamic business processes whose execution, rather than being predetermined at design time, depends on elements of the context in which the process executes. Such elements can include, for example, the type of customer, the geographical location, or the channel.

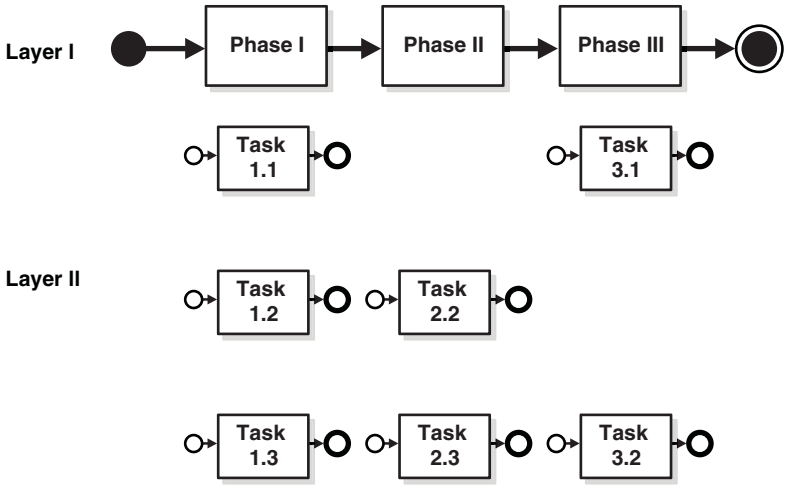
To illustrate further, assume you have an application that performs multichannel banking using various processes. In this scenario, the execution of each process depends on the channel for each particular process instance.

This chapter includes the following sections:

48.1 Introduction to Two-Layer Business Process Management

Two-layer BPM enables you to model business processes using a layered approach. In that model, a first level is a very abstract specification of the business process. Activities of a first-level process delegate the work to processes or services in a second level. Figure 48–1 illustrates this behavior.

Figure 48–1 Two-Layer BPM



In Figure 48–1, the phase I activity of the business process can delegate its work to one of the corresponding layer II processes: Task 1.1, Task 1.2, or Task 1.3.

The two-layer BPM functionality enables you to create the key element (namely, the phase activity) declaratively.

By using the design time and runtime functionality of Oracle Business Rules, you can add more channels dynamically without having to redeploy the business process. Design time at runtime enables you to add rules (columns) to the dynamic routing decision table at runtime. Then, during runtime, business process instances consider those new rules and eventually route the requests to a different channel.

The design time at runtime functionality of Oracle Business Rules also enables you to modify the endpoint reference of a service that is invoked from a phase activity, pointing that reference to a different service.

Note: You can use the design time at runtime functionality of Oracle Business Rules through Oracle SOA Composer and the Oracle Business Rules SDK.

For information about using Oracle SOA Composer and the Oracle Business Rules SDK, see:

- *Oracle Fusion Middleware User's Guide for Oracle Business Rules*
- *Oracle Fusion Middleware Java API Reference for Oracle Business Rules*

To enable two-layer BPM, follow the steps shown in Table 48–1.

Table 48–1 Steps for Enabling Two-Layer BPM

Step	Information
Install Oracle WebLogic Server	*Oracle WebLogic Server Installation Guide*
Design the SOA composite application	Section 48.4.1, "Designing the SOA Composite"

Table 48–1 (Cont.) Steps for Enabling Two-Layer BPM

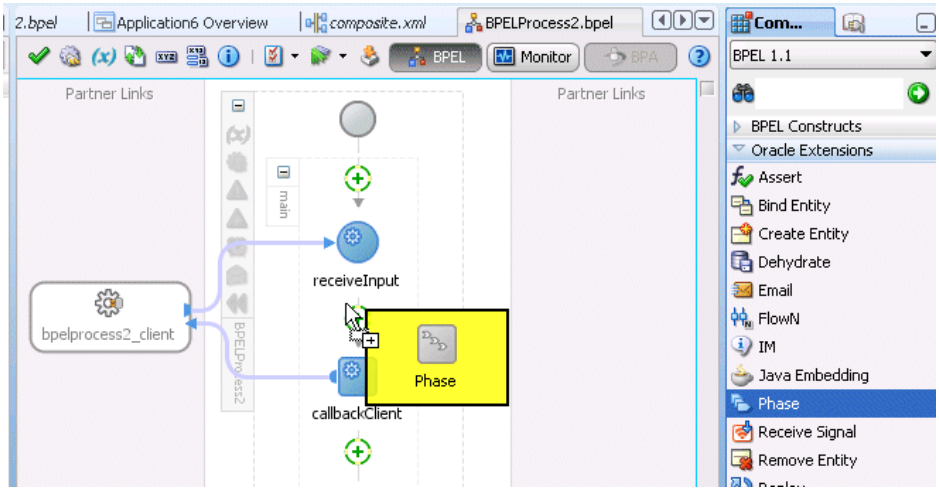
Step	Information
Create element-type variables named `InputPhaseVariable` and `OutputPhaseVar`	Section 48.4.1, "Designing the SOA Composite"
Create a phase activity	Section 48.2, "Creating a Phase Activity"
Create and edit the dynamic routing decision table	Section 48.3, "Creating the Dynamic Routing Decision Table"
Add assign activities to the BPEL process model	Section A.2.2, "Assign Activity"
Create the application deployment profile	Chapter 40, "Deploying SOA Composite Applications"
Create an application server connection	Section 40.7.1.1.1, "Creating an Application Server Connection"
Deploy the application	Chapter 40, "Deploying SOA Composite Applications"

48.2 Creating a Phase Activity

In two-layer BPM, a phase is a level-1 activity in the BPEL process. It complements the existing higher-level Oracle Business Rules and human task BPEL activities.

You add a phase to a process declaratively in Oracle BPEL Designer by dragging and dropping it from the **Oracle Extensions** section of the Component Palette to the process model. Figure 48–2 provides details.

Figure 48–2 Phase Activity in BPEL Designer



Note: The reference WSDL (layer 2 or called references) must have the same abstract WSDL as that for the phase reference that gets automatically created.

48.2.1 How to Create a Phase Activity

You create the phase activity for your composite application after you have created the necessary variables, as described in Section 48.4.1, "Designing the SOA Composite."

To create a phase activity:

1. Double-click the **Phase** activity.
2. In the **Name** field, enter a value.
3. In the **Input and Output Variables** section, select the **Add** icon to add input and output variables.
4. Select **Add Input Variable**. The dialog for selecting a variable appears.
5. Select **Process** > **Variables** > **phaseIn**.
6. Click **OK**. The Phase dialog is displayed with the **phaseIn** variable populated.
7. From the **Input and Output Variables** icon, select **Add Output Variable**. The dialog for selecting a variable appears.
8. Select **Process** > **Variables** > **phaseOut**.
9. Click **OK**. The Phase dialog is displayed with the input and output variable names populated.
10. Click **OK**. The Oracle BPEL Designer displays the BPEL process.
11. From the **File** menu, select **Save All**.
12. Close the BPEL process.
13. Click the **composite.xml** link about the Oracle BPEL Designer. The SOA Composite Editor appears.

48.2.2 What Happens When You Create a Phase Activity

When you create a phase activity, the artifacts described in Table 48–2 are created.

Table 48–2 Artifacts Created with a Phase Activity

Artifact	Description
BPEL scope	At the location where the user dropped the phase activity in the BPEL process, a new BPEL scope is created and inserted into the BPEL process. The scope has the name of the phase activity. Within the scope, several standard BPEL activities are created. The most important ones are one invoke activity to an Oracle Mediator and one receive activity from the Oracle Mediator.
Oracle Mediator component	With the SOA composite application of the BPEL process service component, a new Oracle Mediator service component is created. The Oracle Mediator service component is wired to the phase activity of the BPEL component that comprises the level-1 BPEL process where the phase activity has been dropped into the process model. The input and output of the Oracle Mediator service component is defined by the input and output of the phase activity. The Oracle Mediator plan (the processing instructions of the Oracle Mediator service component) is very simple; it delegates creation of the processing instructions to the Oracle Business Rules service component.

Table 48–2 (Cont.) Artifacts Created with a Phase Activity

Artifact	Description
Oracle Business Rules component	Within the SOA composite application of the BPEL process service component, a new Oracle Business Rules service component is created and wired to the Oracle Mediator component associated with the phase activity of the BPEL process service component. The Oracle Business Rules service component includes a rule dictionary. The rule dictionary contains metadata for such Oracle Business Rules engine artifacts as fact types, rulesets, rules, decision tables, and similar artifacts. As part of creating the Oracle Business Rules service component, the rule dictionary is preinitialized with the following data: ■ Fact Type Model: The data model used for modeling rules. The rule dictionary is populated with a fact type model that corresponds to the input of the phase activity with some fixed data model that is required as part of the contract between the Oracle Mediator and Oracle Business Rules service components. ■ Ruleset: A container of rules used as a grouping mechanism for rules. A ruleset can be exposed as a service. One ruleset is created within the rule dictionary. ■ Decision Table: From an Oracle Business Rules 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 so that the rules can be visualized in a tabular format. The new decision table is created within the ruleset. ■ Decision Service: A decision service is created that exposes the ruleset as a service of the Oracle Business Rules service component. The service interface is used by the Oracle Mediator to evaluate the decision table.

48.2.3 What Happens at Runtime When You Create a Phase Activity

At runtime, the input of the phase activity is used to evaluate the dynamic routing decision table. This is performed by a specific decision component of the phase activity. The result of this evaluation is an instruction for the Oracle Mediator. The Oracle Mediator routes the request to a service based on instructions from the decision component.

Note: In the current release, an asynchronous phase activity is supported. A synchronous or one-way phase activity is not supported.

48.2.4 What You May Need to Know About Creating a Phase Activity

When creating a phase activity, you must know the following:

- Rules that you must either configure or create in the decision service. This is based on data from the payload that you use to evaluate a rule.
- For each rule created in the decision service, you must know the corresponding endpoint URL that must be invoked when a rule evaluates to true. This endpoint URL is used by the Oracle Mediator to invoke the service in layer 2.

  For information on specifying endpoints, see Section 48.4.3, "Creating and Editing the Dynamic Routing Decision Table."

Note: No transformation, assignment, or validation can be performed on a payload.

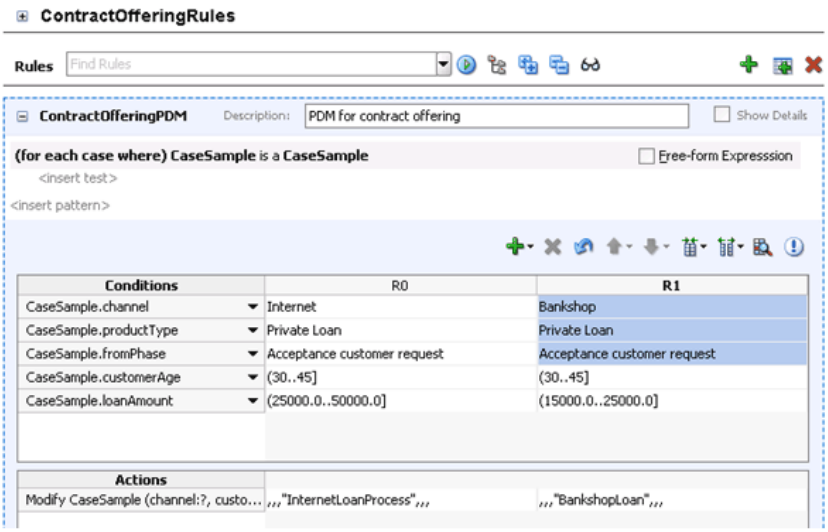
48.3 Creating the Dynamic Routing Decision Table

A Dynamic Routing Decision Table is a decision table evaluated by Oracle Business Rules. Conditions are evaluated on the input data of a phase activity. The result of the evaluation is a routing instruction for the Oracle Mediator.

48.3.1 How to Create the Dynamic Routing Decision Table

After you have created the phase activity, the wizard launches the Oracle Business Rules Designer in Oracle JDeveloper for you to edit the Dynamic Routing Decision Table. Figure 48–3 shows a sample decision table within the Oracle Business Rules Designer.

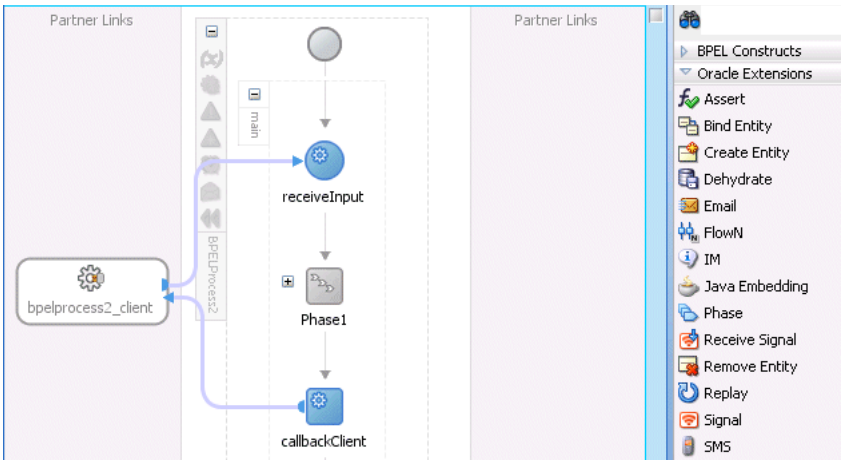
Figure 48–3 Sample Decision Table



You can leave the information for the action attribute **serviceBindingInfo** empty while modeling the level-2 process phases and complete it after the level-1 process is being deployed using Oracle SOA Composer.

Once you have created and edited the Dynamic Routing Decision Table, the new level-1 phase activity appears in the BPEL process in Oracle JDeveloper, as illustrated in Figure 48–4.

Figure 48–4 Completed Level-1 Phase in Oracle JDeveloper



48.3.2 What Happens When You Create the Dynamic Routing Decision Table

By creating the Dynamic Routing Decision Table, you are configuring the decision service to dynamically evaluate the conditions applied to the incoming payload and give the corresponding routing rules to the Oracle Mediator. The Oracle Mediator then executes these rules when invoking the service in layer 2.

More specifically, here is what happens at design time when you create the Dynamic Routing Decision Table:

- A new decision component is created in the composite of the project.
- A new rule dictionary is created in the composite project directory.
- The rule dictionary is populated with a data model that reflects the data model of the phase input; that is, the XML schema of the phase input is imported into the rule dictionary.

48.4 Use Case: Two-Layer BPM

This section describes how to build a sample SOA composite application for routing a customer order.

To build and run the sample:

1. Model the sample by performing these tasks:
   a. Design the SOA composite as described in Section 48.4.1, "Designing the SOA Composite."
   b. Create the phase activity as described in Section 48.4.2, "Creating a Phase Activity."
   c. Create and edit the Dynamic Routing Decision Table as described in Section 48.4.3, "Creating and Editing the Dynamic Routing Decision Table."
   d. Add assign activities to the BPEL process model as described in Section 48.4.4, "Adding Assign Activities to the BPEL Process Model."
2. Deploy the sample with Oracle JDeveloper as described in Section 48.4.5, "Deploying and Testing the Sample."

48.4.1 Designing the SOA Composite

You design the SOA composite application in Oracle JDeveloper.

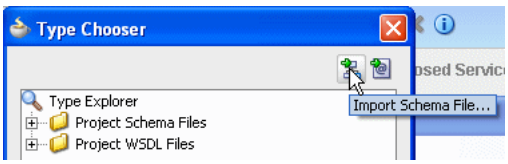
To design the SOA composite:

1. In Oracle JDeveloper, from the **File** menu, select **New**. The New Gallery dialog appears. By default, **Generic Application** is selected.
2. Click **OK**. The Create Generic Application wizard displays the first screen.
3. In the **Application Name** field, enter `BPELPhaseActivity` and then click **Next**. The second page of the Create Generic Application wizard appears.
4. In the **Project Name** field, enter `BPELPhaseCustomerRouter`.
5. In the **Project Technologies** tab, from the **Available** section, select **SOA** and move it to the **Selected** section.
6. Click **Next**. The third page of the Create Generic Application wizard appears.
7. From the **Composite Template** list, select **Composite With BPEL Process**, and click **Finish**.

   The Create BPEL Process dialog appears.
8. In the **Name** field of the Create BPEL Process dialog, enter `CustomerRouterBPELProcess`.
9. From the **Template** list, select **Asynchronous BPEL Process**.
10. Import the schema file (for this example, named **CustomerData.xsd**) into the project **xsd** folder. An XML schema definition (XSD) specifies the types of elements and attributes that may appear in an XML document, their relationship to each other, the types of data they may contain, and other things.
    a. In the **Input** field, click the **Browse Input Elements** icon.

       The Type Chooser dialog displays.
    b. Click the **Import Schema File** icon, as shown in Figure 48–5.

Figure 48–5 Import Schema File Icon



       The Import Schema File dialog displays.
    c. To the right of the **URL** field, click the **Browse Resources** icon.

       The SOA Resource Browser appears.
    d. Select **File System** and, in the **Location** section, search for the schema file (for this example, named **CustomerData.xsd**) in the **artifacts/schema folder**, then click **OK**.
    e. In the Import Schema dialog, ensure the schema file (for this example, named **CustomerData.xsd**) now appears in the **URL** field and the **Copy to Project** option is selected, and then click **OK**.

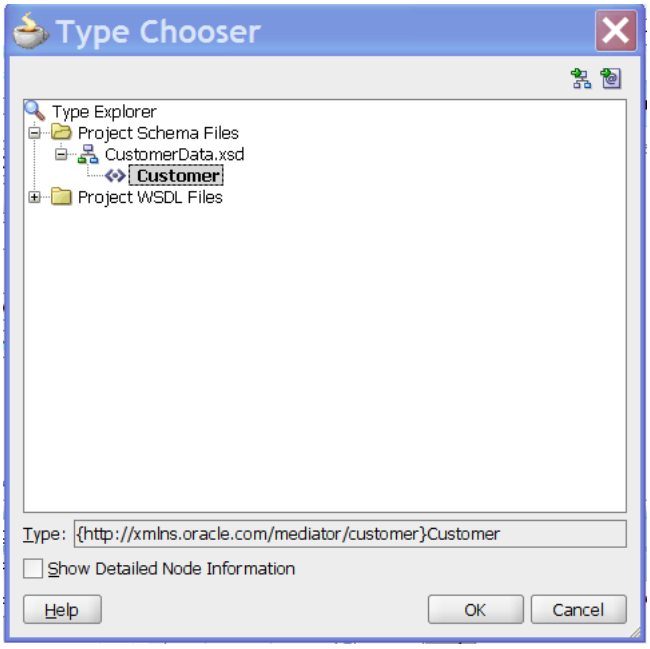
The Localize Files dialog prompts you to import the schema file and any dependent files.

f. Deselect the option **Maintain original directory structure for imported files** and click **OK** to import the files.

The Type Chooser dialog appears.

g. Expand **Project Schema Files** > *schema_file* (for this example, named **CustomerData.xsd**) > **Customer** and then click **OK**, as shown in Figure 48–6.

Figure 48–6 Type Chooser Dialog



11. After importing the schema file, open the **CustomerRouterBPELProcess** BPEL process.

To create variables:

Note: Phase variables can be of the element type only.

1. Click the **Variables** icon. The Variables dialog appears.
2. Click the **Create** icon. The Create Variable dialog appears.
3. In the **Name** field, enter `InputPhaseVariable`.
4. Click the **Element** option.
5. Click the **Browse Elements** icon. The Type Chooser dialog appears.
6. Select **Project Schema Files** > *schema_file* (for this example, named **CustomerData.xsd**) > **Customer**, and then click **OK**. The Create Variable dialog appears with the element name populated.
7. Click **OK**. The Variables dialog is displayed with the variable name populated.
8. Click the **Create** icon in the Variables dialog. The Create Variable dialog appears.

9. In the **Name** field, enter `OutputPhaseVariable`.
10. In the **Type** section, select the **Element** option.
11. Click the **Browse Elements** icon. The Type Chooser dialog appears.
12. Select **Project Schema Files** > ***schema_file*** (for this example, named **CustomerData.xsd**) > **Customer**, and then click **OK**. The Create Variable dialog appears with the element name populated.
13. Click **OK**. The Variables dialog appears with the input and output variable names populated.
14. Click **OK**. The variables have been created and the **CustomerRouterBPELProcess** BPEL process appears.

48.4.2 Creating a Phase Activity

To create a Phase activity:

1. In the **CustomerRouterBPELProcess** BPEL process, drag and drop a phase activity from the Component Palette into the process model, between **receiveInput** and **replyOutput**. The Phase dialog appears.
2. In the **Name** field, enter `CustomerRoutingPhase_1`.
3. From the **Inputs and Outputs Variables** icon, select **Add Input Variable**. The Add Input Variable dialog appears.
4. Select **Process** > **Variables** > ***variable_name*** (for this example, named **phaseIn**), and then click **OK**. The Phase dialog is displayed with the variable populated.
5. From the **Inputs and Outputs Variables** icon, select **Add Output Variable**. The Add Output Variable dialog appears.
6. Select **Process** > **Variables** > ***variable_name*** (for this example, named **OutputPhaseVar**).
7. Click **OK**. The Phase dialog displays the input and output variable names.
8. Click **OK**. The **CustomerRouterBPELProcess** BPEL process appears.
9. From the **File** menu, select **Save All**.
10. Close the **CustomerRouterBPELProcess** BPEL process.
11. Click **composite.xml**. The SOA Composite Editor is displayed.

Note:

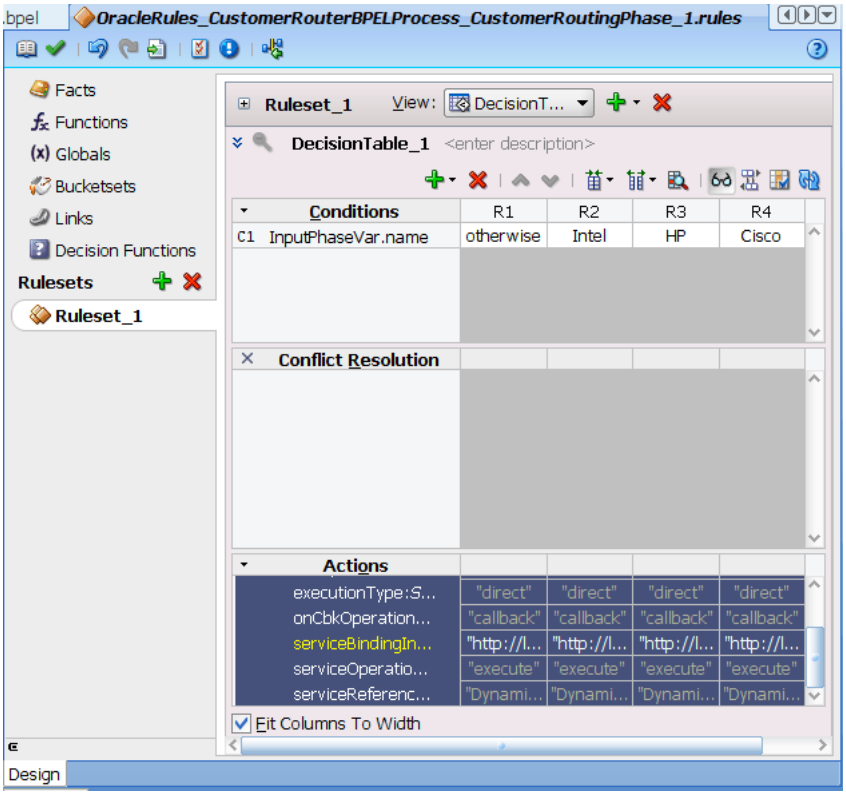
- As part of the phase activity wizard, three components are created: Oracle Business Rules, Oracle Mediator, and Dynamic Reference.
- The Oracle Business Rules service component returns an executable case for the Oracle Mediator service component, because of the rules defined.
- The Oracle Mediator service component performs routing based on the routing rules received from the Oracle Business Rules service component.
- The Dynamic Reference component is the dummy reference for the second-level processes.
- The rule dictionary is populated with the fact type model of the Oracle Mediator and the fact type corresponding to the input of the phase activity, which in this case is CustomerData.
- An empty decision table called the Routing Table is created that must be edited to provide dynamic routing rules.

48.4.3 Creating and Editing the Dynamic Routing Decision Table

To create and edit the Dynamic Routing Decision Table:

1. Open the **CustomerRouterBPELProcess** BPEL process, and double-click the **Phase** activity in the process diagram. The Phase dialog appears.
2. Click the **Edit Dynamic Rules** button. The Oracle Business Rules Designer page appears.
3. Under **Rulesets**, click **Ruleset_1**. The Ruleset_1 page with an empty Routing Table appears, as shown in Figure 48–7.

Figure 48–7 Ruleset Page



4. In **DecisionTable_1**, click the **Add** icon, then **Action** > **Assert New**. The **Actions** section of the table appears.
5. In the **serviceBindingInfo**, specify the SOAP endpoints, replacing the hostname and host port with SOA server details. In this example, `localhost` is the host server and `8001` is the host port. The composite in this example is named `CustomerRouter` and it must already be deployed.
   - In the **otherwise** column, enter the following:

     `http://`*`hostname`*`:`*`host_port`*`/soa-infra/services/default/CustomerRouter!1.0/DefaultCustomerRouterService`

   - In the **Intel** column, enter the following:

     `http://`*`hostname`*`:`*`host_port`*`/soa-infra/services/default/CustomerRouter!1.0/SilverCustomerRouterService`

   - In the **Cisco** column, enter the following:

     `http://`*`hostname`*`:`*`host_port`*`/soa-infra/services/default/CustomerRouter!1.0/GoldCustomerRouterService`

   - In the **HP** column, enter the following:

     `http://`*`hostname`*`:`*`host_port`*`/soa-infra/services/default/CustomerRouter!1.0/PlatinumCustomerRouterService`

48.4.4 Adding Assign Activities to the BPEL Process Model

Before deploying the phase activity, you must initialize the phase variables. You do this by adding assign activities in the phase in the BPEL process.

To add assign activities to the BPEL process model:

1. Click the **CustomerRouterBPELProcess** BPEL process.
2. Drag and drop an **Assign** activity from the Component Palette into the process model between the **receiveInput** activity and the **Phase** activity. The **Assign** activity is added to the process model.
3. Double-click the **Assign** activity. The Assign dialog is displayed.
4. Select the **General** tab.
5. In the **Name** field, enter `AssignInput`.
6. Select the **Copy Rules** tab.
7. Select the source section. For this example, **Variables** > **Process** > **Variables** > **inputVariable** > **payload** > **ns1:Customer** is selected.
8. Select the target section. For this example, **Variables** > **Process** > **Variables** > **inputVariable** > **payload** > **ns1:Customer** is selected.
9. Drag the source node to the target node (for example, drag the source **ns1:Customer** node to the target **ns1:Customer** node).

   The input copy rule is recorded at the bottom of the Edit Assign dialog, as shown in Table 48–3.

Table 48–3 Input Copy Rule for Adding Assign Activities

From	To
inputVariable/payload//ns1:Customer	**InputPhaseVar///payload/ns1:Customer**

10. Click **OK**. The **CustomerRouterBPELProcess** process is displayed again.
11. Drag and drop another **Assign** activity from the Component Palette into the process model between the **Phase** activity and the **replyOutput** activity. The new **Assign** activity is added to the process model.
12. Double-click the **Assign** activity. The Assign dialog appears.
13. In the **Name** field in the **General** tab, enter `AssignOutput`.
14. Select the **Copy Rules** tab.
15. Select the source section. For this example, **Variables** > **Process** > **Variables** > **OutputPhaseVar** > **payload** > **ns1:Customer/ns1:status** is selected.
16. Select the target section. For this example, **Process** > **Variables** > **outputVariable** > **payload** > **client:processResponse** > **client:result** is selected.
17. Drag the source **ns1:status** node to the target **client:result** node.

    The output copy rule is recorded, as shown in Table 48–4.

Table 48–4 Output Copy Rule for Adding Assign Activities

From	To
OutputPhaseVar///ns1:Customer/ns1:status	**outputVariable/payload//client:processResponse/client:result**

18. Click **OK**. The **CustomerRouterBPELProcess** BPEL process appears after the input and output assign activities are created.
19. From the **File** menu, select **Save All**.

48.4.5 Deploying and Testing the Sample

For instructions on deploying the sample, see Section 40.7, "Deploying SOA Composite Applications."

For instructions on testing a composite instance in Oracle Enterprise Manager Fusion Middleware Control, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

49

Integrating the Spring Framework in SOA Composite Applications

This chapter describes how to use the spring framework to integrate components that use Java interfaces into SOA composite applications. Oracle SOA Suite uses the spring framework functionality provided by the WebLogic Service Component Architecture (SCA) of Oracle WebLogic Server. This chapter also describes how to integrate components that use Java interfaces with components that use WSDL files in the same SOA composite application.

This chapter includes the following sections:

For more information about the WebLogic SCA functionality used by Oracle SOA Suite, see *Oracle Fusion Middleware Developing WebLogic SCA Applications for Oracle WebLogic Server*.

49.1 Introduction to the Spring Service Component

The spring framework is a lightweight container that makes it easy to use different types of services. Lightweight containers can accept any JavaBean, instead of specific types of components.

WebLogic SCA enables you to use the spring framework to create Java applications using plain old Java objects (POJOs) and expose components as SCA services and references. In SCA terms, a WebLogic spring framework SCA application is a collection of POJOs plus a spring SCA context file that wires the classes with SCA services and references.

You can use the spring framework to create service components and wire them within a SOA composite application using its dependency injection capabilities. SCA can extend spring framework capabilities as follows:

- Publish spring beans as SCA component services that can be accessed by other SCA components or by remote clients
- Provide spring beans for service references wired to services of other components

Like all service components, spring components have a `componentType` file. The interfaces defined in the `componentType` file use the `interface.java` definition to identify their service and reference interfaces.

Services are implemented by beans and are targeted in the spring context file. References are supplied by the runtime as implicit (or virtual) beans in the spring context file.

You can also integrate Enterprise JavaBeans with SOA composite applications through use of Java interfaces (with no requirement for SDO parameters). For information, see Chapter 35, "Integrating Enterprise JavaBeans with SOA Composite Applications."

For more information about the spring framework, visit the following URL:

`http://www.osoa.org/display/Main/SCA+and+Spring+Framework`

49.2 Integration of Java and WSDL-Based Components in the Same SOA Composite Application

In releases before 11*g* Release 1 11.1.1.3, components in SOA composite applications were entirely WSDL-based. Starting with 11*g* Release 1 11.1.1.3, you can integrate components using Java interfaces and WSDL files in a SOA composite application in the SOA Composite Editor. As an example, this integration enables a spring service component to invoke an Oracle BPEL Process Manager or an Oracle Mediator service component to invoke an Enterprise JavaBean, and so on.

The following types of component integrations are supported:

- Java components to WSDL components

  If you drag a wire from a Java interface (for example, Enterprise JavaBeans service or spring service component) to a component that does not support Java interfaces (for example, Oracle Mediator, Oracle BPEL Process Manager, or others) a compatible WSDL is generated for the component interfaces.

- WSDL components to Java components

  If you drag a wire from a WSDL interface to a component that does not support WSDL files (for example, a spring service component), a compatible Java interface is automatically generated. It is also possible to wire an existing WSDL interface to an existing Java interface. In this case, there is no checking of the compatibility between the WSDL and Java interfaces. You must ensure that it is correct.

- Java components to Java components

  If you create a spring service component, you can automatically configure it with Java interface-based EJB service and reference binding components. No WSDL files are required.

49.2.1 Java and WSDL-Based Integration Example

When wiring any two service components (or a service component with a binding component), each end of the wire has an interface defined. With XML, those interfaces must have the same WSDL definition, and are defined with `interface.wsdl` in the `composite.xml` file or *`component`*`.componentType` file.

From the JAX-WS point of view, when wiring a Java interface (which is defined by `interface.java`) to a WSDL interface, it is assumed that the two interfaces are compatible. This is typically enforced and automated by Oracle JDeveloper.

> **Note:** *Only* use Oracle JDeveloper to create and modify the `composite.xml`, `componentType`, and spring context files described in this section. Do *not* directly edit these files in **Source** view. These examples are provided to show you how Java interfaces and WSDL files are integrated in a SOA composite application. Use of Oracle JDeveloper to achieve this functionality is described in subsequent sections of this chapter.

For example, assume you have a Java interface for a service, as shown in Example 49–1.

Example 49–1 Java Interface for a Service

```
public interface PortfolioService {
  public double getPorfolioValue(String portfolioId);
}
```

Assume the implementation can use an additional `StockQuote` service that is implemented by another component that may be a BPEL process, an external web service, or an EJB. Example 49–2 provides details.

Example 49–2 Additional Java Interface for a Service

```
public interface StockQuote {
  public double getQuote (String symbol);
}
```

The `componentType` file for the spring framework lists the `PortfolioService` service and the `StockQuote` service with the `interface.java` definitions. Example 49–3 provides details.

Example 49–3 componentType File

```
<componentType xmlns="http://xmlns.oracle.com/sca/1.0">
    <service name="PortfolioService ">
        <interface.java interface="com.bigbank.PortfolioService"/>
    </service>
    <reference name="StockService">
        <interface.java interface="com.bigbank.StockQuote"/>
    </reference>
</componentType>
```

The implementation class implements the service interface and provides a setter for the reference interface. Example 49–4 provides details.

Example 49–4 Implementation of the Service Interface

```
public class PortfolioServiceImpl implements PortfolioService {
  StockQuote stockQuoteRef;

  public void setStockService (StockQuote ref) {
    stockQuoteRef = ref;
  }
```

```
  public double getPorfolioValue(String portfolioId) {
    //--  use stock service
    //--  return value
  }
}
```

The spring context file calls out the services and references and binds them to the implementation. Example 49–5 provides details.

Example 49–5 Callout of Services and References by the Spring Context

```
<beans ...>
  <sca:service  name="PortfolioService" type="com.bigbank.PortfolioService"
 target="impl">
  </sca:service>

  <sca:reference  name="StockService" type="com.bigbank.StockQuote">
  </sca:reference>

  <bean id ="impl" class ="com.bigbank.PortfolioServiceImpl">
     <property name="stockService" ref="StockService"/>
  </bean>
</beans>
```

The `composite.xml` file of the composite defines the components and references. Example 49–6 provides details.

Example 49–6 Definition of Components and References in the composite.xml File

```
<composite ...>
    <import location="PortfolioService.wsdl" />
    <service name="PortfolioService">
        <interface.wsdl
 interface="http://bigbank.com/#wsdl.interface(PortfolioService)" />
        <binding.ws
 port="http://bigbank.com/#wsdl.endpoint(PortfolioService/PortfolioServicePort)"/>
    </service>
    <wire>
        <source.uri>PortfolioService</source.uri>
        <target.uri>PortfolioComp/PortfolioService</target.uri>
    </wire>
    <component name="PortfolioComp">
        <implementation.spring src="spring-context.xml"/>
    </component>
    <wire>
        <source.uri>PortfolioService/StockService</source.uri>
        <target.uri>StockService</target.uri>
    </wire>
    <reference name="StockService">
        <interface.java interface="com.bigbank.StockQuote"/>
        <binding.ejb uri="StockService#com.bigbank.StockQuote"/>
    </reference>
</composite>
```

49.2.2 Using Callbacks with the Spring Framework

Oracle SOA Suite uses callbacks for both `interface.wsdl` and `interface.java`. However, the concept of callbacks does not exist in the spring framework. For Oracle SOA Suite services and references, a callback is specified (in the metadata) as a second port type for `interface.wsdl` or a second Java name for `interface.java`. The

spring metadata has only `sca:services` and `sca:references` and no way to specify a callback.

To design a callback with spring, you must provide `sca:services` and `sca:references` with a specific name. If you create both a `sca:service` and `sca:reference` using the naming conventions of *some*`Service` and *some*`ServiceCallback`, Oracle SOA Suite recognizes this convention and creates a single service or reference with a callback.

For example, assume you create the syntax shown in Example 49–7 in the spring context file with the spring editor in Oracle JDeveloper:

Example 49–7 Callbacks with the Spring Service Component

```
<sca:service  name="StockService"
 type="oracle.integration.platform.blocks.java.callback.StockService"
 target="impl" />

<sca:reference  name="StockServiceCallback"
 type="oracle.integration.platform.blocks.java.callback.StockServiceReply" />
```

Oracle SOA Suite automatically creates a single service (in the spring `componentType` file) as shown in Example 49–8:

Example 49–8 Single Service

```
    <service name="StockService">
      <interface.java
 interface="oracle.integration.platform.blocks.java.callback.StockService"

callbackInterface="oracle.integration.platform.blocks.java.callback.StockServiceRe
ply"/>
    </service>
```

In the SOA Composite Editor, if a spring `interface.java` with a callback interface is dragged to a WSDL component (for example, Oracle BPEL Process Manager, Oracle Mediator, or others), a WSDL with two port types is generated (technically, a wrapper WSDL, which is a WSDL that imports two other WSDLs, each having a single port type).

If you drag a WSDL or Java interface that has a callback to a spring service component, a single interface is displayed in the SOA Composite Editor. However, inside the spring editor, you find both a `sca:service` and `sca:reference` that have the same naming conventions (*some*`Service` and *some*`ServiceCallback`).

49.3 Creating a Spring Service Component in Oracle JDeveloper

This section describes how to create a spring service component and wire the component as follows in Oracle JDeveloper:

- To Java interface-based EJB services and references (Java-to-Java integration)
- To an Oracle Mediator service component (Java-to-WSDL integration)

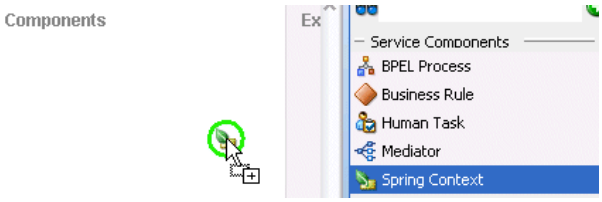
For an overview of spring service component integration in the Fusion Order Demo, see Section 49.6, "Spring Service Component Integration in the Fusion Order Demo."

49.3.1 How to Create a Spring Service Component in Oracle JDeveloper

To create a spring service component in Oracle JDeveloper:

1. From the Component Palette, drag a **Spring Context** service component into the SOA Composite Editor, as shown in Figure 49–1.

Figure 49–1 Spring Context Service Component

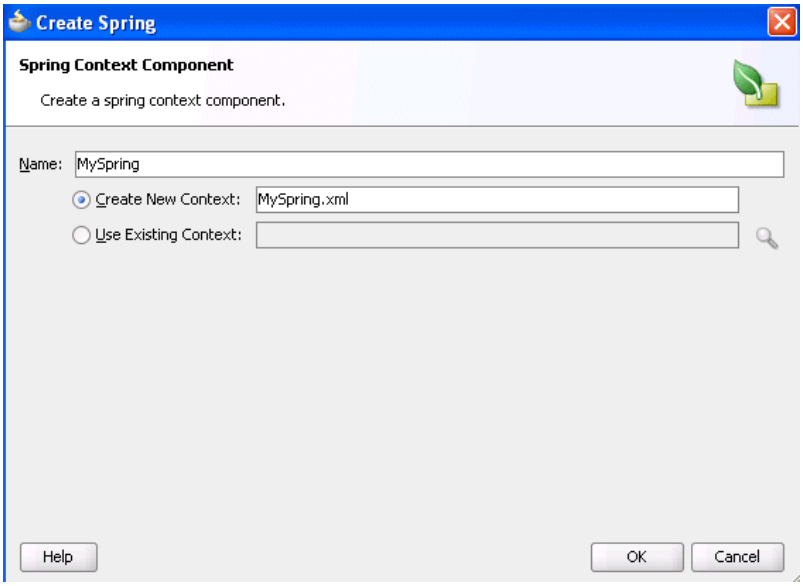


The Create Spring dialog is displayed.

2. In the **Name** field, enter a name for the spring service component. The name becomes both the component name and the spring context file name. Figure 49–2 provides details.

   You can also select **Use Existing Context** and click **Browse** to select an existing spring file. For example, you may want to import a spring context that was created in Oracle JDeveloper, but outside of Oracle SOA Suite. If you browse and select a spring context from another project, it is copied to the SOA project.

Figure 49–2 Create Spring Dialog



Note: A standalone spring version of WebLogic SCA is also available for use. This version is typically used outside of Oracle SOA Suite. This version is accessible by selecting **Spring 2.5 JEE** from the Component Palette while inside the spring editor.

3. Click **OK**.

A spring icon is displayed in the SOA Composite Editor.

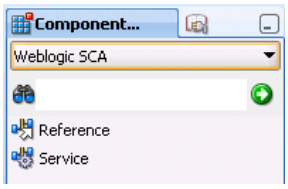
4. Double-click the icon to display the contents of the spring context in the spring editor.

```
<?xml version="1.0" encoding="windows-1252" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:lang="http://www.springframework.org/schema/lang"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:sca="http://xmlns.oracle.com/weblogic/weblogic-sca"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
 http://www.springframework.org/schema/util
 http://www.springframework.org/schema/util/spring-util-2.5.xsd
 http://www.springframework.org/schema/aop
 http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
 http://www.springframework.org/schema/jee
 http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
 http://www.springframework.org/schema/lang
 http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
 http://www.springframework.org/schema/tx
 http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
 http://www.springframework.org/schema/tool
 http://www.springframework.org/schema/tool/spring-tool-2.5.xsd
 http://xmlns.oracle.com/weblogic/weblogic-sca META-INF/weblogic-sca.xsd">
  <!--Spring Bean defintions go here-->
</beans>
```

5. From the Component Palette, select **Weblogic SCA** from the dropdown list.

   The list is refreshed to display the selections shown in Figure 49–3.

Figure 49–3 WebLogic SCA Menu



6. Drag a **Service** icon into the spring editor.

   The Insert Service dialog appears.

7. Complete the fields shown in Table 49–1 to define the target bean and Java interface.

Table 49–1 Insert Service Dialog

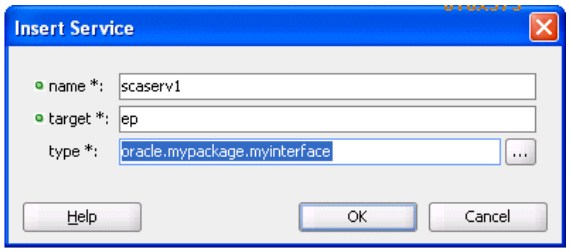
Field	Description
name	Enter a name.
target	Enter the target bean. This action enables you to expose the bean as a service. **Note**: Ensure that this target exists. There is no validation support that checks for the existence of this target.

Table 49–1 (Cont.) Insert Service Dialog

Field	Description
type	Enter the Java interface.

When complete, the Insert Service dialog looks as shown in Figure 49–4.

Figure 49–4 Insert Service Dialog



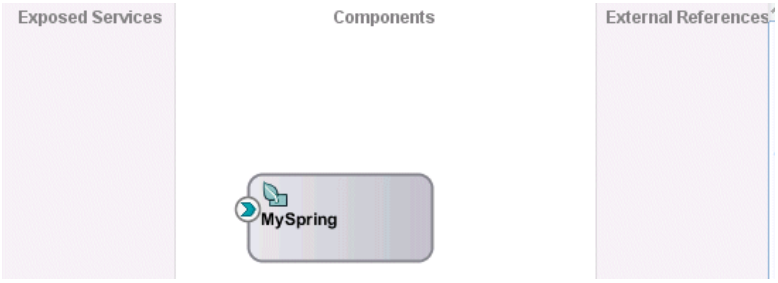
8. Click **OK**.

   The target bean becomes the service interface in the spring context.

```
<?xml version="1.0" encoding="windows-1252" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:lang="http://www.springframework.org/schema/lang"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:sca="http://xmlns.oracle.com/weblogic/weblogic-sca"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
 http://www.springframework.org/schema/util
 http://www.springframework.org/schema/util/spring-util-2.5.xsd
 http://www.springframework.org/schema/aop
 http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
 http://www.springframework.org/schema/jee
 http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
 http://www.springframework.org/schema/lang
 http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
 http://www.springframework.org/schema/tx
 http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
 http://www.springframework.org/schema/tool
 http://www.springframework.org/schema/tool/spring-tool-2.5.xsd
 http://xmlns.oracle.com/weblogic/weblogic-sca META-INF/weblogic-sca.xsd">
  <!--Spring Bean defintions go here-->
  <sca:service name="scaserv1" target="ep"
type="oracle.mypackage.myinterface"/>
</beans>
```

Note that if you close the spring editor and return to the SOA Composite Editor, you see that a handle has been added to the left side of the spring service component, as shown in Figure 49–5.

Figure 49–5 Service Handle



9. Return to the spring editor.
10. Drag a **Reference** icon from the list shown in Figure 49–3 into the spring editor.

    The Insert Reference dialog is displayed.
11. Complete the dialog, as shown in Table 49–2, and click **OK**.

Table 49–2 Insert Reference Dialog

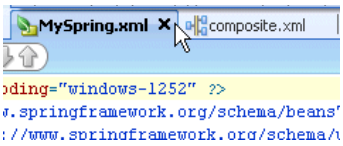
Field	Description
name	Enter a name.
type	Enter the Java interface.

When complete, the spring context displays the service and reference in the spring editor.

```
<?xml version="1.0" encoding="windows-1252" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:lang="http://www.springframework.org/schema/lang"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:sca="http://xmlns.oracle.com/weblogic/weblogic-sca"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
 http://www.springframework.org/schema/util
 http://www.springframework.org/schema/util/spring-util-2.5.xsd
 http://www.springframework.org/schema/aop
 http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
 http://www.springframework.org/schema/jee
 http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
 http://www.springframework.org/schema/lang
 http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
 http://www.springframework.org/schema/tx
 http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
 http://www.springframework.org/schema/tool
 http://www.springframework.org/schema/tool/spring-tool-2.5.xsd
 http://xmlns.oracle.com/weblogic/weblogic-sca META-INF/weblogic-sca.xsd">
  <!--Spring Bean defintions go here-->
  <sca:service name="scaserv1" target="ep"
type="oracle.mypackage.myinterface"/>
  <sca:reference name="scaref1" type="external.bean.myInterface"/>
</beans>
```

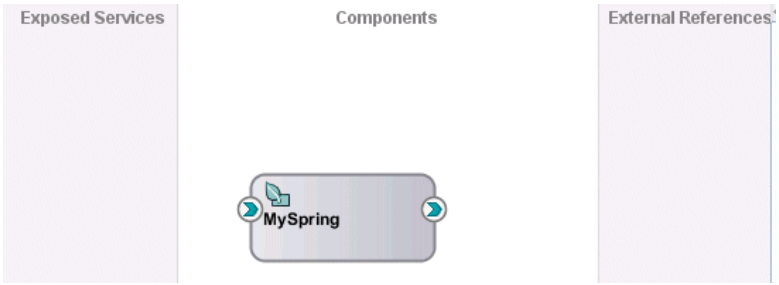
12. Close the spring context file, as shown in Figure 49–6.

Figure 49–6 Spring Context File



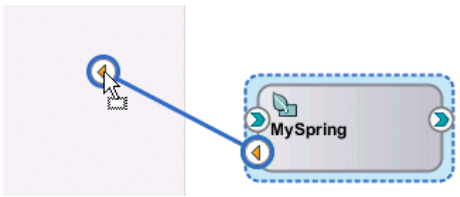
Note that a handle is added to the right side of the spring service component, as shown in Figure 49–7.

Figure 49–7 Reference Handle



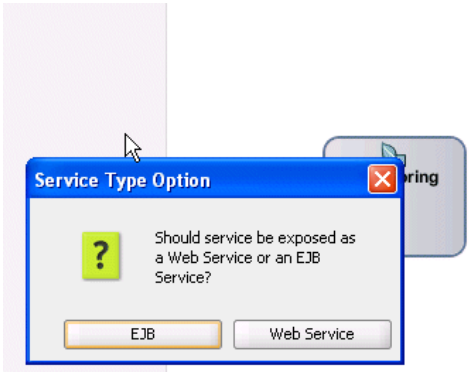
13. Drag the left handle into the **Exposed Services** swimlane to create a service binding component, as shown in Figure 49–8.

Figure 49–8 Service Binding Component



You are prompted to select to expose the service as either a web service or as an EJB service, as shown in Figure 49–9.

Figure 49–9 Service Type To Create

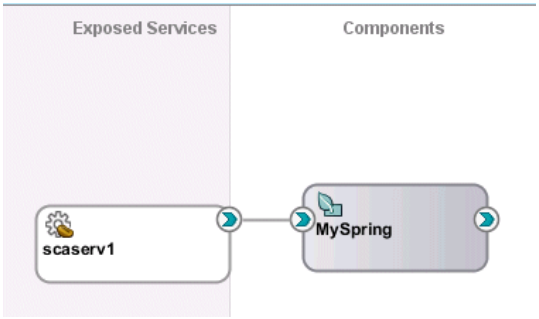


- **EJB**: This exposes the EJB service through a Java interface; this selection does not require the use of a WSDL file.

- **Web Service**: This exposes the web service through a SOAP WSDL interface. If you select this option, a WSDL is generated from the Java Interface for compatibility with the spring service component.

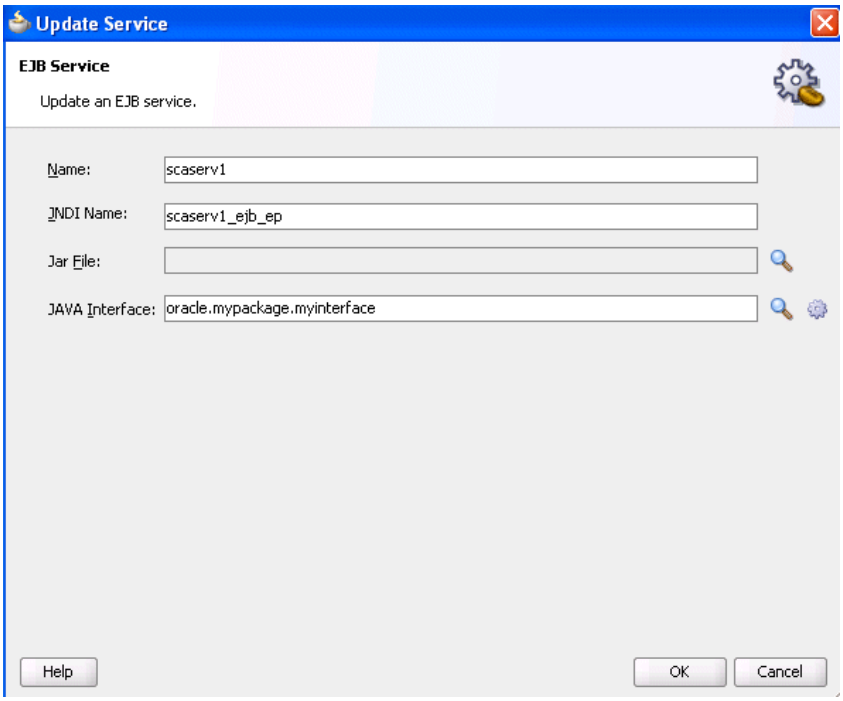
14. Select to expose this service as either an EJB or Web service. A service is automatically created in the **Exposed Services** swimlane and wired to the spring service component (for this example, **EJB** was selected). Figure 49–10 provides details.

Figure 49–10 EJB Service Binding Component Wired to the Spring Service Component



15. Double-click the EJB service to display the automatically completed configuration, as shown in Figure 49–11. The configuration details were created from the values you entered in the Insert Service dialog in Step 7.

Figure 49–11 EJB Service Dialog in Exposed Services Swimlane



16. Replace the default JNDI name that was automatically generated with the name applicable to your environment.

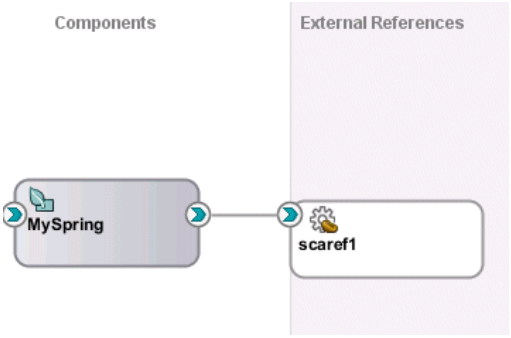
17. Close the dialog.

18. Drag the right handle of the spring service component into the **External References** swimlane to create a reference binding component.

    You are prompted with the same spring type option message as shown in Step 13.

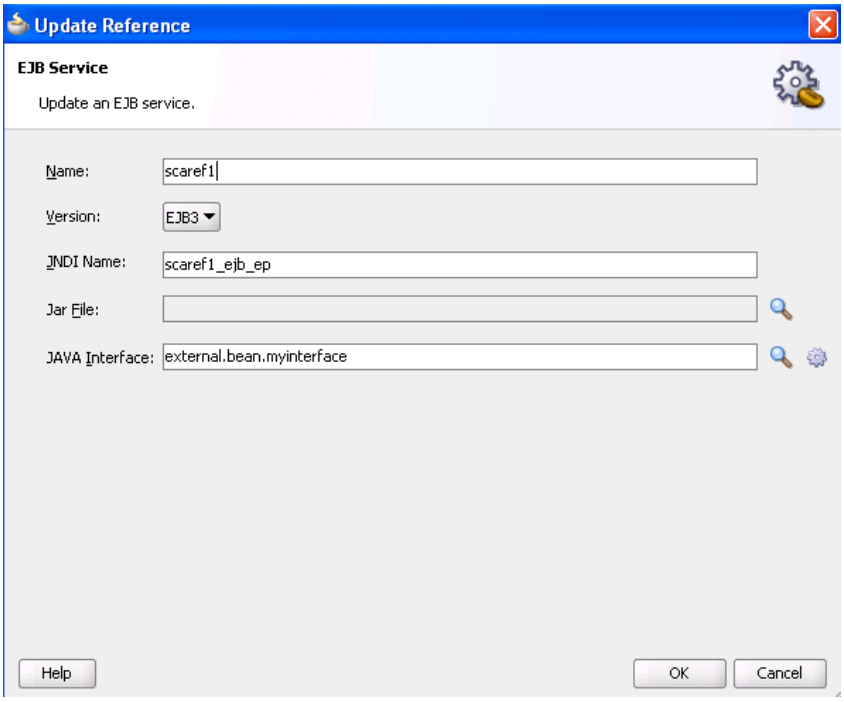
19. Select an option to expose this reference. A reference is automatically created in the **External References** swimlane and wired to the spring service component (for this example, **EJB** was selected). Figure 49–12 provides details.

Figure 49–12 EJB Reference Binding Component Wired to the Spring Service Component



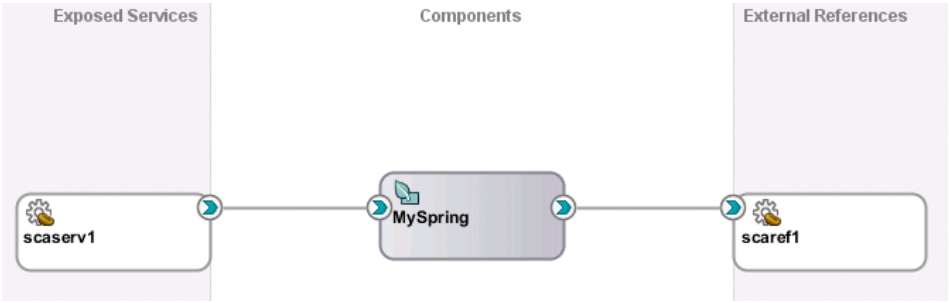
20. Double-click the EJB reference to display the automatically completed configuration, as shown in Figure 49–13. The configuration details were created from the values you entered in the Insert Reference dialog in Step 11.

Figure 49–13 EJB Reference Dialog in External References Swimlane



21. Close the dialog and return to the SOA Composite Editor, as shown in Figure 49–14.

Figure 49–14 Java Interface-Based EJB Service and Reference Binding Components



22. Place the cursor over both the right handle of the service (as shown in Figure 49–15) and the left handle of the spring service component (as shown in Figure 49–16). The Java interface is displayed.

Figure 49–15 Java Interface of Service

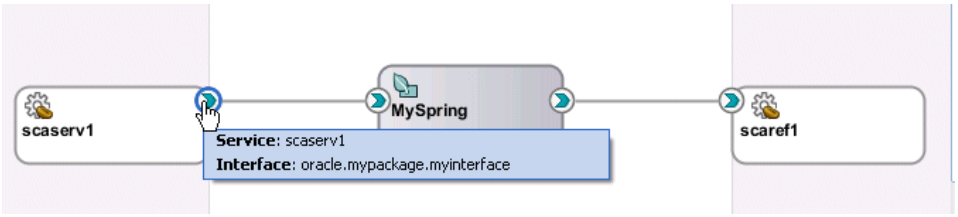
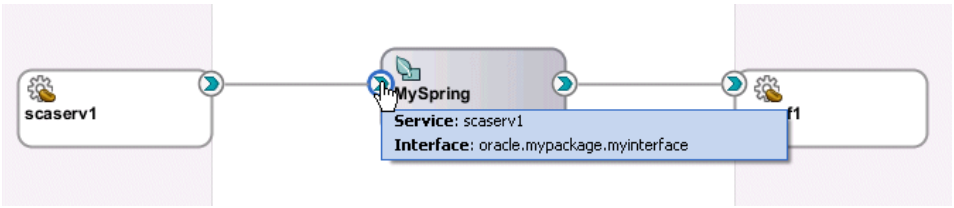


Figure 49–16 Java Interface of Spring Service Component



23. Perform the same action on the right handle of the spring service component and the left handle of the reference binding component to display its Java interface.

24. If you want to view the interfaces for the spring service component in the `componentType` file, select this file in the Application Navigator. The interfaces for both components are defined by `interface.java`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version 1.0 at [2/27/10 1:13 PM]. -->
<componentType
                xmlns="http://xmlns.oracle.com/sca/1.0"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:ui="http://xmlns.oracle.com/soa/designer/">
  <service name="scaserv1">
    <interface.java interface="oracle.mypackage.myinterface"/>
  </service>
  <reference name="scaref1">
    <interface.java interface="external.bean.myInterface"/>
  </reference>
</componentType>
```

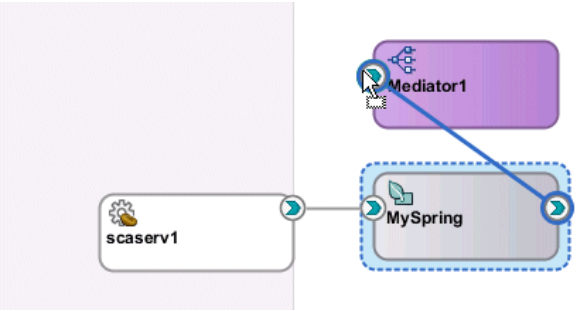
25. In the Application Navigator, select the `composite.xml` file to display similar details.

```
<service name="scaserv1">
   <interface.java interface="oracle.mypackage.myinterface"/>
   <binding.ejb uri="scaserv1_ejb_ep" ejb-version="EJB3"/>
 </service>
 <component name="MySpring">
   <implementation.spring src="MySpring.xml"/>
 </component>
 <reference name="scaref1">
   <interface.java interface="external.bean.myInterface"/>
   <binding.ejb uri="scaref1_ejb_ep" ejb-version="EJB3"/>
 </reference>
 <wire>
   <source.uri>orderprocessor_client_ep</source.uri>
   <target.uri>OrderProcessor/orderprocessor_client</target.uri>
 </wire>
 <wire>
   <source.uri>scaserv1</source.uri>
   <target.uri>MySpring/scaserv1</target.uri>
 </wire>
 <wire>
   <source.uri>MySpring/scaref1</source.uri>
   <target.uri>scaref1</target.uri>
 </wire>
</composite>
```

26. If you wire the right handle of the spring service component to an XML-based component such as Oracle Mediator instead of the Java interface-based EJB reference, a compatible WSDL file is generated. The following steps provide details.

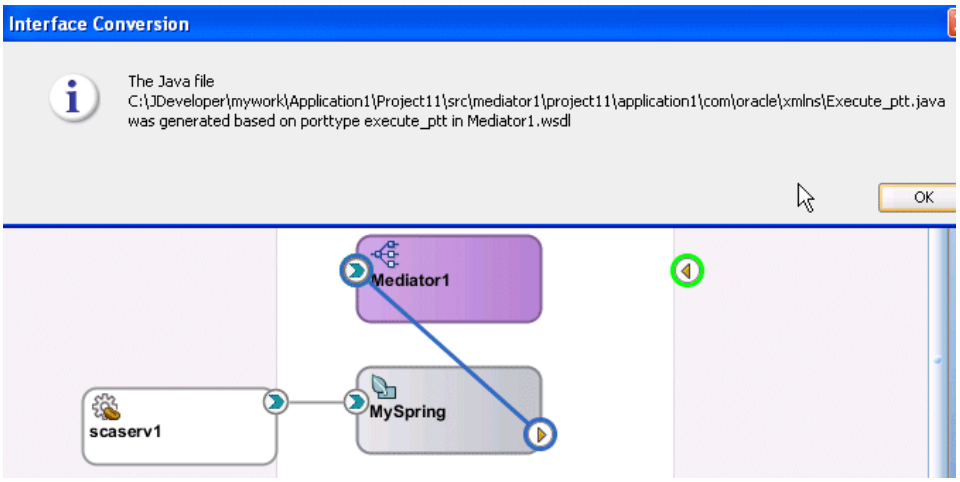
    a. Drag the right handle of the spring service component to the Oracle Mediator, as shown in Figure 49–17.

Figure 49–17 Integration of Spring Service Component and Oracle Mediator



    b. Click **OK** when prompted to acknowledge that a compatible interface was created from the Oracle Mediator WSDL file.

Figure 49–18 Java File Creation from the Oracle Mediator WSDL File



If you drag a wire between a Java interface and a WSDL-based component, and the WSDL file with the default name (based on the Java Interface name) already exists, you are prompted with four options. Click **Cancel** to cancel creation of the wire. Figure 49–19 provides details.

Figure 49–19 Existing WSDL File

c. Place the cursor over both the right handle of the spring service component (as shown in Figure 49–20) and the left handle of the Oracle Mediator (as shown in Figure 49–21) to display the compatible interface.

Figure 49–20 Spring Service Component Interface

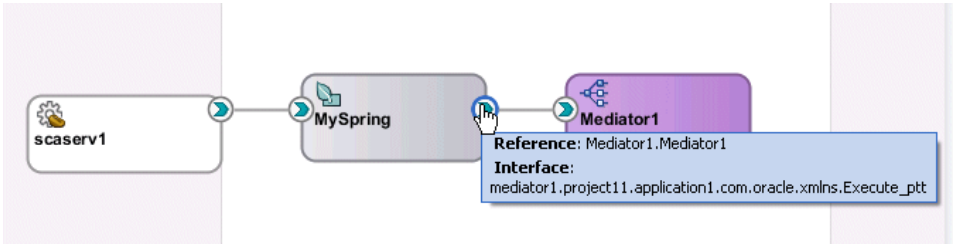
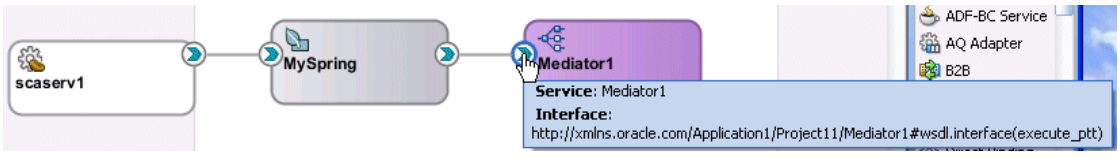


Figure 49–21 Oracle Mediator Interface



d. Double-click the spring service component to display the contents of the spring context file in the spring editor.

```
<?xml version="1.0" encoding="windows-1252" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:lang="http://www.springframework.org/schema/lang"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:sca="http://xmlns.oracle.com/weblogic/weblogic-sca"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-2.5.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
        http://www.springframework.org/schema/tool
        http://www.springframework.org/schema/tool/spring-tool-2.5.xsd
        http://xmlns.oracle.com/weblogic/weblogic-sca
META-INF/weblogic-sca.xsd">
  <!--Spring Bean defintions go here-->
  <sca:service name="scaserv1" target="ep"
type="oracle.mypackage.myinterface"/>
  <sca:reference
type="mediator1.project1.application4.com.oracle.xmlns.Execute_
ptt" name="Mediator1.Mediator1"/>
</beans>
```

For more information about integrating components that use Java interfaces with components that use WSDL files in the same SOA composite application, see Section 49.6, "Spring Service Component Integration in the Fusion Order Demo."

Notes:

- When integrating a component that uses a Java interface with a component that uses a WSDL file in the SOA Composite Editor, if a specific interface class is not found in the classpath, but the source file does exist in the SOA project, the JAR files in the `SCA-INF/lib` directory of the current loader are automatically refreshed to discover the interface class.
- You can also create BPEL process partner links with services that uses Java interfaces. You select this type of service in the Service Explorer dialog when creating a partner link. For more information, see Section 4.3, "Introduction to Partner Links."

49.3.2 What You May Need to Know About Java Class Errors During Java-to-WSDL Conversions

When a Java-to-WSDL conversion fails because of a bad Java class and you modify the Java code to correct the problem, you must restart Oracle JDeveloper. Not doing so results in a Java-to-WSDL conversion failure because the new class does not get reloaded.

49.4 Defining Custom Spring Beans Through a Global Spring Context

You can define custom spring beans through a global spring context definition. This configuration enables you to define these beans only once, at the global level.

49.4.1 How to Define Custom Spring Beans Through a Global Spring Context

To define custom spring beans through a global spring context:

1. Add the custom spring bean definitions into the following file:

   SOA_HOME/soa/modules/oracle.soa.ext_11.1.1/classes/springse-extension-global-beans.xml

2. Add the corresponding classes in either the `lib` directory (as a JAR file) or the classes directory (as extracted files of the JAR file).

   SOA_HOME/soa/modules/oracle.soa.ext_11.1.1/lib | classes

For more information, see the `readme.txt` file located in the following directory:

SOA_HOME/soa/modules/oracle.soa.ext_11.1.1

> **Note:** A server restart is required to pick up newly added spring beans.

49.5 Using the Predefined Spring Beans

Oracle SOA Suite provides the following predefined spring beans:

- `headerHelperBean`: For getting and setting header properties.
- `instanceHelperBean`: For getting the following information:
  - The instance ID of the composite instance currently running.
  - The instance ID of the component instance currently running.
  - The composite distinguished name (DN) containing the component.
  - The name of the spring service component.
- `loggerBean`: For providing context-aware logging messages.

The predefined spring beans are automatically injected into the spring service component. However, you must explicitly integrate the predefined spring beans into a SOA composite application by providing a reference to the bean in the spring context file.

For an example of how to reference `loggerBean` and `headerHelperBean` in a spring context file, see Section 49.5.4, "How to Reference Predefined Spring Beans in the Spring Context File."

49.5.1 IHeaderHelperBean.java Interface for headerHelperBean

Example 49–9 shows the `IHeaderHelperBean.java` interface for the `headerHelperBean` bean.

Example 49–9 IHeaderHelperBean.java Interface

```
package oracle.soa.platform.component.spring.beans;
/**
 * Interface for getting and setting header properties.
 * These properties will be set on the normalized message - and passed on
 * to the respective reference that the local reference is wired to on
 * composite level.
 * <br/>
 * In order to use this bean from within your context, declare property
 * with ref="headerHelperBean". E.g.
 * &lt;property name="headerHelper" ref="<b>headerHelperBean</b>"/>
 */
public interface IHeaderHelperBean
{
    /**
     * Get a property from the normalized message header. Note that these
     * properties are defined, and are the same ones, one can get/set via
     * mediator or bpel process
     * @param pKey the property key, case sensitive
     * @return the value, or null in case not found
     */
    public String getHeaderProperty (String pKey);
    /**
     * Set a property on the normalized message header. Note that these
     * properties are defined, and are the same ones, one can get/set via
     * mediator or bpel process
     * @param pKey the property key, case sensitive
     * @param pValue the value to be set
     */
    public void setHeaderProperty (String pKey, String pValue);
}
```

49.5.2 IInstanceHelperBean.java Interface for instancerHelperBean

Example 49–10 shows the `IInstanceHelperBean.java` interface for the `instanceHelperBean` bean.

Example 49–10 IInstanceHelperBean.java Interface

```
package oracle.soa.platform.component.spring.beans;

import oracle.integration.platform.instance.engine.ComponentInstanceContext;
/**
 * Instancehelper Bean, gives access to composite / component + instance
 information
 * <br/>
 * In order to use this bean from within your context, declare property
 * with ref="instanceHelperBean". E.g.
 * &lt;property name="instanceHelper" ref="<b>instanceHelperBean</b>"/>
 */
public interface IInstanceHelperBean
{
  /**
   * Returns the instance id of the composite instance currently running
```

```
   * @return the composite instance id
   */
  public String getCompositeInstanceId ();

  /**
   * Returns the instance id of the component instance currently running
   * @return the component instance id
   */
  public String getComponentInstanceId ();

  /**
   * Returns the composite dn containing this component
   * @return the composite dn
   */
  public String getCompositeDN ();

  /**
   * Returns the name of this spring component
   * @return the component name
   */
  public String getComponentName ();

}
```

49.5.3 ILoggerBean.java Interface for loggerBean

Example 49–11 shows the `ILoggerBean.java` interface for the `loggerBean` bean.

Example 49–11 ILoggerBean.java Interface

```
package oracle.soa.platform.component.spring.beans;

import java.util.logging.Level;

/**
 * Logger bean interface, messages will be logged as
 * [&lt;composite instance id>/&lt;component instance id>] &lt;message>
 * <br/>
 * In order to use this bean from within your context, declare property
 * with ref="loggerBean". E.g.
 * &lt;property name="logger" ref="<b>loggerBean</b>"/>
 * @author clemens utschig
 */
public interface ILoggerBean
{

  /**
   * Log a message, with Level.INFO
   * @param message
   */
  public void log (String message);

  /**
   * Log a message with desired level
   * @param pLevel the log level
   * @param message the message to log
   */
  public void log (Level pLevel, String message);

  /**
```

```
   * Log a throwable with the desired level
   * @param level the level to log with
   * @param message the message
   * @param th the exception (throwable) to log
   */
  public void log (Level level, String message, Throwable th);

}
```

49.5.4 How to Reference Predefined Spring Beans in the Spring Context File

You create references to the predefined beans in the spring context file.

To reference predefined spring beans in the spring context file:

1. Open the spring context file in **Source** view in Oracle JDeveloper.
2. Add references to the `loggerBean` and `headerHelperBean` predefined beans.

```
<?xml version="1.0" encoding="windows-1252" ?>
. . .
. . .
  <!--
    The below sca:service(s) corresponds to the services exposed by the
    component type file: SpringPartnerSupplierMediator.componentType
  -->

  <!-- expose the InternalPartnerSupplierMediator + EJB as service

    <service name="IInternalPartnerSupplier">
      <interface.java
 interface="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>
    </service>

  -->
  <sca:service name="IInternalPartnerSupplier"
        target="InternalPartnerSupplierMediator"

type="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>

  <!-- expose the InternalPartnerSupplierMediator + Mock as service

    <service name="IInternalPartnerSupplierSimple">
      <interface.java
 interface="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>
    </service>

  -->
  <sca:service name="IInternalPartnerSupplierSimple"
        target="InternalPartnerSupplierMediatorSimple"

type="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>

  <!-- the partner supplier mediator bean with the mock ep -->
  <bean id="InternalPartnerSupplierMediatorSimple"

class="com.otn.sample.fod.soa.internalsupplier.InternalSupplierMediator"
        scope="prototype">
        <!-- inject the external partner supplier bean -->
        <property name="externalPartnerSupplier"
```

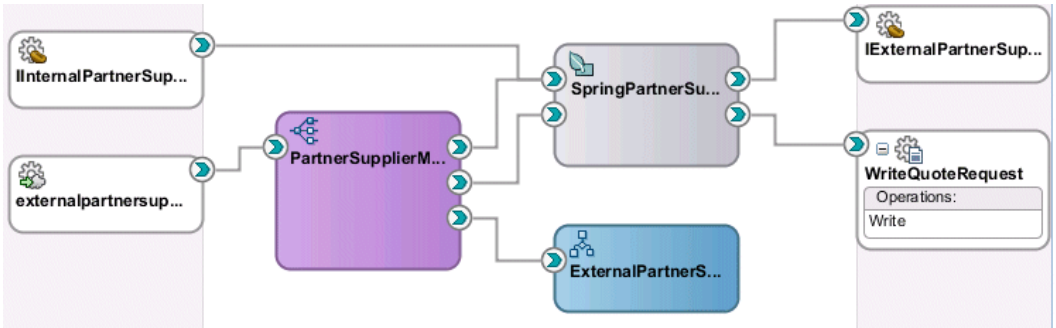
```
            ref="IExternalPartnerSupplierServiceMock"/>
        <!-- inject the quoteWriter -->
        <property name="quoteWriter" ref="WriteQuoteRequest"/>
        <!-- context aware logger, globally available bean [ps3] -->
        <property name="logger" ref="loggerBean"/>
        <!-- headerHelper bean -->
        <property name="headerHelper" ref="headerHelperBean"/>
  </bean>

  <!-- the partner supplier mediator bean with the ejb -->
  <bean id="InternalPartnerSupplierMediator"

class="com.otn.sample.fod.soa.internalsupplier.InternalSupplierMediator"
        scope="prototype">
        <!-- inject the external partner supplier bean -->
        <property name="externalPartnerSupplier"
            ref="IExternalPartnerSupplierService"/>
        <!-- inject the quoteWriter -->
        <property name="quoteWriter" ref="WriteQuoteRequest"/>
        <!-- context aware logger, globally available bean [ps3] -->
        <property name="logger" ref="loggerBean"/>
        <!-- headerHelper bean -->
        <property name="headerHelper" ref="headerHelperBean"/>
  </bean>
. . .
. . .
```

This syntax is included in the spring context file of the Partner Supplier Composite application of the Fusion Order Demo. For more information about the Fusion Order Demo, see Section 49.6, "Spring Service Component Integration in the Fusion Order Demo."

49.6 Spring Service Component Integration in the Fusion Order Demo

The Partner Supplier Composite application of the Fusion Order Demo demonstrates how the spring service component obtains a price quote from a partner warehouse. Figure 49–22 shows the SOA Composite Editor for this composite application.

Figure 49–22 Partner Supplier Composite with Spring Service Component



IInternalPartnerSupplier is exposed as an external client service in the **Exposed Services** swimlane.

The Oracle Mediator service component **PartnerSupplierMediator** routes client requests differently based on the amount of the quote:

- Quotes below $2000 are routed to Oracle BPEL Process Manager.

- Requests that are more than $2000 and less than $3000 are routed to the **SpringPartnerSupplierMediator** spring service component. An external EJB reference binding component **IExternalPartnerSupplierService** is invoked to obtain a quote. An external file adapter **WriteQuoteRequest** is invoked for writing the quote results to a file.
- Requests greater than $3000 are routed to the **SpringPartnerSupplierMediator** spring service component. However, these requests are not routed to the external EJB reference binding component. Instead they are handled internally by implementing the EJB interface. The external file adapter **WriteQuoteRequest** is also invoked for writing the quote results to a file.

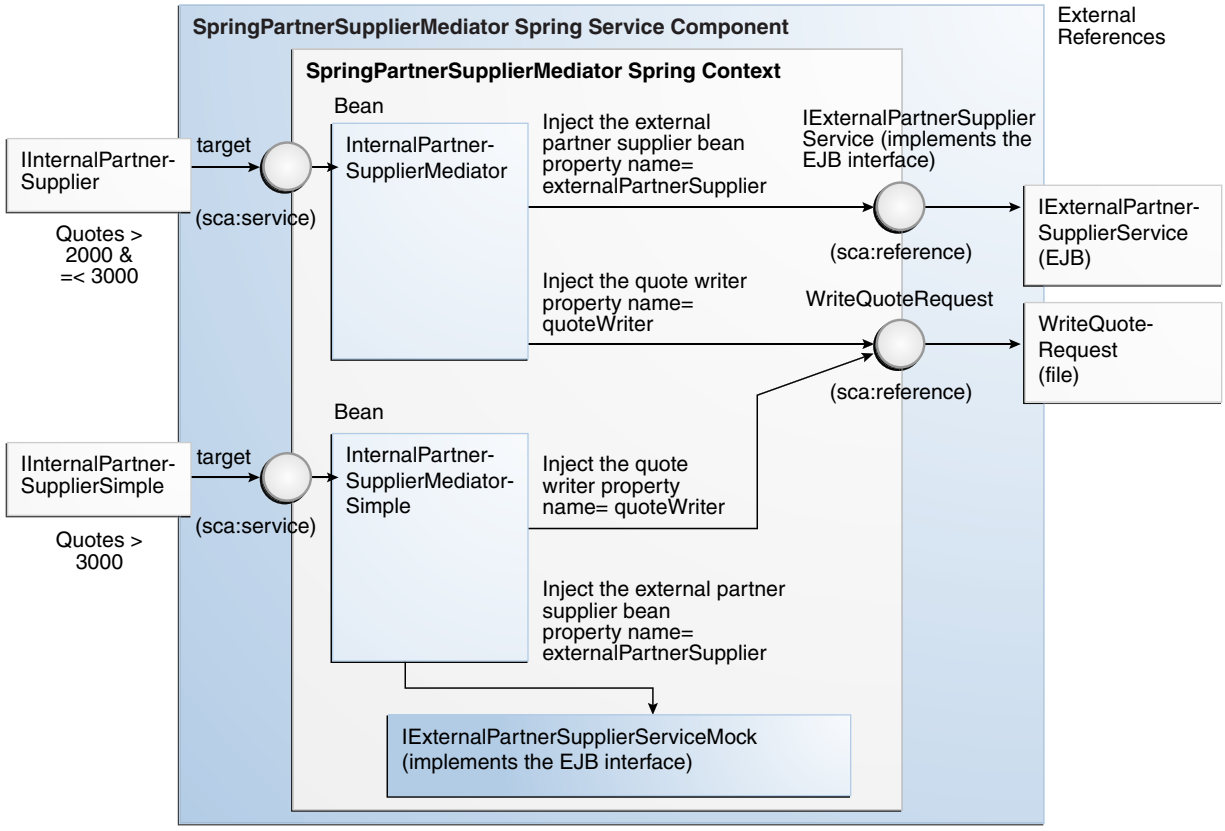
Figure 49–23 provides an overview of this behavior.

For requests that are more than $2000 and less than $3000, the target bean **InternalPartnerSupplierMediator** is exposed as a service. The Java interface **IInternalPartnerSupplier** is used. In the **External References** swimlane, the Java interface **IExternalPartnerSupplierService** is exposed as an external EJB for obtaining a quote.

For requests that are more than $3000, the target bean **InternalPartnerSupplierMediatorSimple** is exposed as a service. The Java interface **IInternalPartnerSupplier** is used. The internal Java Interface **IExternalPartnerSupplierServiceMock** is used to obtain a quote. The **IExternalPartnerSupplierService** reference in the **External References** swimlane is *not* invoked.

In the **External References** swimlane, since the **WriteQuoteRequest** reference uses a WSDL-based file adapter and does not support Java interfaces, a compatible WSDL file is generated.

Figure 49–23 Spring Architecture in Fusion Order Demo



Example 49–12 shows the `IInternalPartnerSupplier.java` file. `IInternalPartnerSupplier` is implemented by `InternalSupplierMediator`.

Example 49–12 IInternalPartnerSupplier.java

```
package com.otn.sample.fod.soa.internalsupplier;
import
 com.otn.sample.fod.soa.internalsupplier.exception.InternalSupplierException;
import java.util.List;
/**
 * The interface for the spring based service, with a typed list.
 *
 *                 !!! ATTENTION !!!
 * This interface was used to generate the wsdl
 *  (IInternalPartnerSupplierService.wsdl) - DO NOT MODIFY!
 *
 */
public interface

IInternalPartnerSupplier
{
    /**
     * Get a price for a list of orderItems
     * @param pOrderItems the list of orderitems
     * @return the price
     */
    public double getPriceForOrderItemList(List<Orderitem> pOrderItems)
```

```
        throws InternalSupplierException;

}
```

The `SpringPartnerSupplierMediator.componentType` file in Example 49–13 shows the services and references defined for the spring service component shown in Figure 49–23.

Example 49–13 SpringPartnerSupplierMediator.componentType File

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version 1.0 at [7/16/09 2:36 PM]. -->
<componentType
               xmlns="http://xmlns.oracle.com/sca/1.0"
               xmlns:xs="http://www.w3.org/2001/XMLSchema"
               xmlns:ui="http://xmlns.oracle.com/soa/designer/">
  <service name="IInternalPartnerSupplier">
    <interface.java
 interface="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>
  </service>
  <service name="IInternalPartnerSupplierSimple">
    <interface.java
 interface="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>
  </service>
  <reference name="IExternalPartnerSupplierService">
    <interface.java
 interface="com.otn.sample.fod.soa.externalps.IExternalPartnerSupplierService"/>
  </reference>
  <reference name="WriteQuoteRequest">
    <interface.java
interface="writequoterequest.partnersuppliercomposite.weblogicfusionorderdemo.file
.adapter.pcbpel.com.oracle.xmlns.Write_ptt"/>
  </reference>
</componentType>
```

Example 49–14 shows the `SpringPartnerSupplierMediator.xml` spring context file.

Example 49–14 SpringPartnerSupplierMediator.xml spring context File

```
<?xml version="1.0" encoding="windows-1252" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:util="http://www.springframework.org/schema/util"
      xmlns:jee="http://www.springframework.org/schema/jee"
      xmlns:lang="http://www.springframework.org/schema/lang"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xmlns:sca="http://xmlns.oracle.com/weblogic/weblogic-sca">
       <!--
   The below sca:service(s) corresponds to the services exposed by the
   component type file: SpringPartnerSupplierMediator.componentType
 -->

<!-- expose the InternalPartnerSupplierMediator + EJB as service

   <service name="IInternalPartnerSupplier">
     <interface.java
 interface="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>
   </service>
```

```
 -->
 <sca:service name="IInternalPartnerSupplier"
       target="InternalPartnerSupplierMediator"
       type="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>

 <!-- expose the InternalPartnerSupplierMediator + Mock as service

   <service name="IInternalPartnerSupplierSimple">
     <interface.java
 interface="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>
   </service>

 -->
 <sca:service name="IInternalPartnerSupplierSimple"
       target="InternalPartnerSupplierMediatorSimple"
       type="com.otn.sample.fod.soa.internalsupplier.IInternalPartnerSupplier"/>

 <!-- the partner supplier mediator bean with the mock ep -->

<bean id="InternalPartnerSupplierMediatorSimple"
       class="com.otn.sample.fod.soa.internalsupplier.InternalSupplierMediator"
       scope="prototype">

       <!-- inject the external partner supplier bean -->
       <property name="externalPartnerSupplier"
           ref="IExternalPartnerSupplierServiceMock"/>
       <!-- inject the quoteWriter -->

<property name="quoteWriter" ref="WriteQuoteRequest"/>
       <!-- context aware logger, globally available bean [ps3] -->
       <property name="logger" ref="loggerBean"/>
<!-- headerHelper bean -->
       <property name="headerHelper" ref="headerHelperBean"/>
 </bean>
 <!-- the partner supplier mediator bean with the ejb -->

<bean id="InternalPartnerSupplierMediator"
       class="com.otn.sample.fod.soa.internalsupplier.InternalSupplierMediator"
       scope="prototype">
       <!-- inject the external partner supplier bean -->
       <property name="externalPartnerSupplier"
           ref="IExternalPartnerSupplierService"/>
       <!-- inject the quoteWriter -->
<property name="quoteWriter" ref="WriteQuoteRequest"/>
       <!-- context aware logger, globally available bean [ps3] -->
       <property name="logger" ref="loggerBean"/>
<!-- headerHelper bean -->
       <property name="headerHelper" ref="headerHelperBean"/>
 </bean>

 <!-- mock bean for the IExternalPartnerSupplierService -->
 <bean id="IExternalPartnerSupplierServiceMock"

 class="com.otn.sample.fod.soa.externalps.test.MockExternalPartnerSupplierTest"/>

<!--
   Use a reference from the outside world based on the
   IExternalPartnerSupplierService interface.
   The below is specified on the SpringPartnerSupplierMediator.componentType -
   and wired to an external EJB binding.
```

```
      <reference name="IExternalPartnerSupplierService">
     <interface.java
 interface="com.otn.sample.fod.soa.externalps.IExternalPartnerSupplierService"/>
   </reference>   -->
<sca:reference name="IExternalPartnerSupplierService"
       type="com.otn.sample.fod.soa.externalps.IExternalPartnerSupplierService"/>
        <!--
     <reference name="WriteQuoteRequest">
       <interface.java

interface="writequoterequest.partnersuppliercomposite.weblogicfusionorderdemo.file
.adapter.pcbpel.com.oracle.xmlns.Write_ptt"/>
     </reference>   -->
 <sca:reference

type="writequoterequest.partnersuppliercomposite.weblogicfusionorderdemo.file.adap
ter.pcbpel.com.oracle.xmlns.Write_ptt"
               name="WriteQuoteRequest"/>
</beans>
```

For information on downloading and installing the Fusion Order Demo and using the Partner Supplier Composite, see Section 3.2, "Setting Up the Fusion Order Demo Application."

After download, see the following Fusion Order Demo directory for Java code samples used by the Partner Supplier Composite:

```
CompositeServices\PartnerSupplierComposite\src\com\otn\sample\fod\soa
```

49.6.1 How to Use EJBs with Java Vector Type Parameters

Your Java code may include vectors. However, vectors cannot be serialized to XML without declaring the content POJOs. The following example provides an overview of how to resolve this issue and uses code samples from the Fusion Order Demo.

To use EJBs with Java vector type parameters:

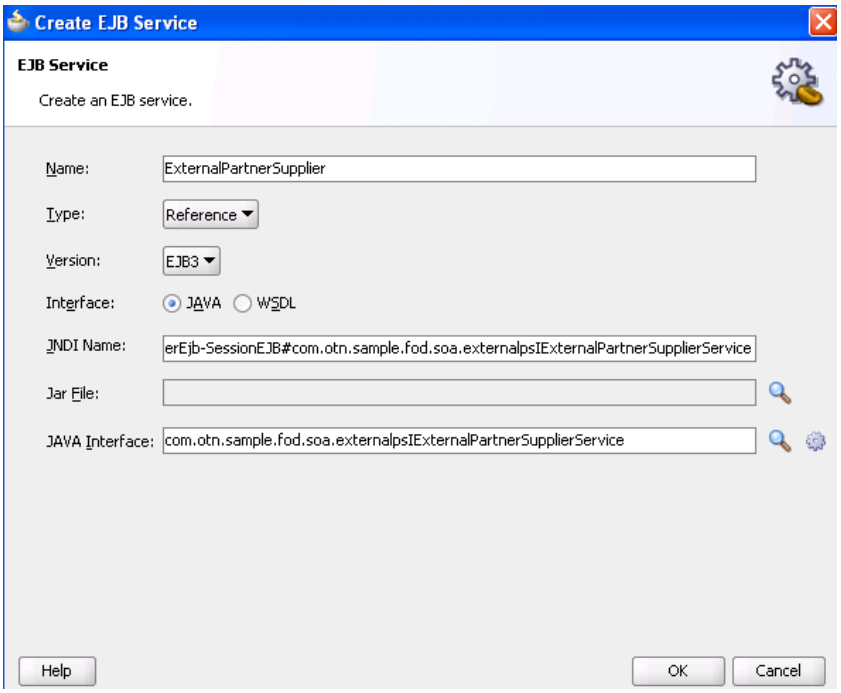
1. Assume your Java code includes vectors, as shown in Figure 49–24.

Figure 49–24 Vectors

```
 */
package com.otn.sample.fod.soa.externalps;

import ...;

public interface IExternalPartnerSupplierService
{
    /**
     * Get the quote for a set of order lines
     * @param pOrderLines the order lines via untyped vector, the perfect case
     * to use java for mediation
     * @return a somewhat "complex" PriceQuote object
     * @throws ExternalSupplierException in case an unknown object has been
     *  passed
     */
    public PriceQuote getQuoteForItems (Vector pOrderLines)
        throws ExternalSupplierException;
```

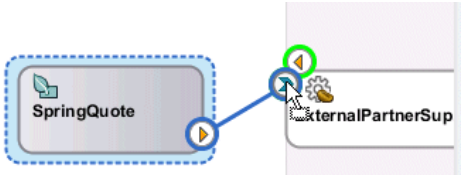
2. Create an EJB binding reference based on the Java interface class and the JNDI name. Figure 49–25 provides an example.

Figure 49–25 EJB Binding Reference Creation



3. Wire the EJB reference to the spring service component, as shown in Figure 49–26.

Figure 49–26 EJB Reference Wired to Spring Service Component



A new reference is created in the spring context file. Figure 49–27 provides details.

Figure 49–27 Reference Addition to Spring Context File

```
<sca:service name="InternalSupplierService" target="InternalSupplierBean"
         type="com.otn.sample.fod.soa.internalsupplier.IInternalSupplier"
         xmlns="http://www.springframework.org/schema/sca"/>
<bean name="InternalSupplierBean" scope="prototype"
  class="com.otn.sample.fod.soa.internalsupplier.InternalSupplierImpl"/>
<sca:reference type="com.otn.sample.fod.soa.externalps.IExternalPartnerSupplierService"
               name="ExternalPartnerSupplier"/>
```

4. Enable spring to inject the reference into the class by declaring a public member of type `IExternalPartnerSupplierService`. Figure 49–28 provides details.

Figure 49–28 Public Member Declaration

```
package com.otn.sample.fod.soa.internalsupplier;

import ...;

public class InternalSupplierImpl implements IInternalSupplier
{
    /**
     * Get an instance of the partner supplier injected
     */
    public IExternalPartnerSupplierService externalPartnerSupplier = null;


    /**
     * Get a quote for a list of items
     * @param pOrderItems the list of orderItems
     * @return the price for the list
     */
    public double getQuoteForOrder (List <Orderitem> pOrderItems)
    {
        double price = 0.0;
        if (pOrderItems == null)
            return price;

        for (Orderitem item : pOrderItems)
        {
            price = price + (item.getPrice() * item.getQuantity());
        }
        return price;
    }
}
```

5. Add a property with the name of the member to `IExternalPartnerSupplierService` and refer to the `ExternalPartnerSupplier` reference bean. Figure 49–29 provides details.

Figure 49–29 Property Added with Name of the Member

```
    class="com.otn.sample.fod.soa.internalsupplier.InternalSupplierImpl">
        <property name="externalPartnerSupplier" ref="ExternalPartnerSupplier"/>
  </bean>
  <sca:reference type="com.otn.sample.fod.soa.externalps.IExternalPartnerSupplierService"
                 name="ExternalPartnerSupplier"/>
</beans>
```

This converts the vectors to EJB parameters.

49.7 JAXB and OXM Support

Oracle Fusion Middleware provides support for using Java Architecture for XML Binding (JAXB) and the EclipseLink O/X-Mapper (OXM) to map Java classes to XML data. You can store and retrieve data in memory in any XML format without implementing a specific set of XML routines for the program's class structure. This support enables you to perform the following:

- Map Java objects to XML data
- Map XML data back to Java objects

For design information about external metadata for JAXB mappings, visit the following URL:

`http://wiki.eclipse.org/EclipseLink/DesignDocs/277920`

For information about JAXB OXM and the OXM mapping file (`eclipselink-oxm.xsd`), visit the following URLs:

`http://wiki.eclipse.org/EclipseLink/FAQ/WhatIsMOXy`

`http://wiki.eclipse.org/EclipseLink/Examples/MOXy`

`http://wiki.eclipse.org/Category:XML`

You can also map Java classes to XML data when integrating Enterprise JavaBeans with SOA composite applications. For more information, see Chapter 35, "Integrating Enterprise JavaBeans with SOA Composite Applications."

49.7.1 Extended Mapping Files

Oracle SOA Suite extends JAXB and OXM file support through use of an extended mapping (EXM) file. If an EXM file is present in the class path of the design time project, then it can be used for Java-to-WSDL conversions. The EXM file provides data binding metadata in the following situations:

- When you cannot add the JAXB annotations into the Java source and must specify them separately
- When scenarios are not covered by JAXB (for example, with top level elements like method return types or parameter types)

The external JAXB annotations can be specified either directly in the EXM file or included in the separate TopLink JAXB mapping OXM file that can be referred to from the EXM file.

Oracle SOA Suite design time supports placing the EXM file in either the source path (`SCA-INF/src`) or the class path (`SCA-INF/classes` or a JAR in `SCA-INF/lib`).

Placing the EXM file in the source path (`SCA-INF/src`) enables you to edit the EXM using Oracle JDeveloper (files in the class path do not appear in the Application Navigator in Oracle JDeveloper). When project compilation is complete, the EXM file (and any XML files that it imports) is copied to the class path (`SCA-INF/classes`) for deployment.

If the EXM file is in the source path, it must still be in the same corresponding directory structure.

Example 49–15 and Example 49–16 provide examples of EXM files.

Example 49–15 EXM Sample File

```
<java-web-service-endpoint
          xmlns="http://xmlns.oracle.com/weblogic/weblogic-wsee-databinding"
          xmlns:oxm="http://www.eclipse.org/eclipselink/xsds/persistence/oxm"
          databinding="toplink.jaxb">

  <xml-schema-mapping>
       <toplink-oxm-file
 java-package="weblogic.wsee.databinding.internal.test.toplink"
 file-path="./person-oxm.xml"/>
  </xml-schema-mapping>

<!--
  <web-service name="hello-ws" target-namespace="hello-ns"/>
  <java-methods>
  <java-method name="hello" oxm:xml-mixed="false">
```

```
    <oxm:xml-elements>
       <oxm:xml-element type="java.lang.Integer"/>
    </oxm:xml-elements>
    <web-result name="result"/>
    <java-params>
        <java-param oxm:xml-mixed="false">
           <oxm:xml-elements>
             <oxm:xml-element type="java.lang.String"/>
           </oxm:xml-elements>
           <web-param name="request"/>
        </java-param>
    </java-params>
  </java-method>
  </java-methods>
  -->
</java-web-service-endpoint>
```

Example 49–16 EXM Sample File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<java-wsdl-mapping
 name="weblogic.wsee.databinding.internal.test.toplink.CollectionMapExtTypeArg"
           xmlns="http://xmlns.oracle.com/weblogic/weblogic-wsee-databinding"
           xmlns:oxm="http://www.eclipse.org/eclipselink/xsds/persistence/oxm"
           databinding="toplink.jaxb">
  <soap-binding parameter-style="BARE"/>
  <java-methods>
  <java-method name="testListOfCustomer">
     <java-params>
         <java-param>
            <oxm:xml-element
 type="weblogic.wsee.databinding.internal.test.toplink.Customer"/>
         </java-param>
     </java-params>
  </java-method>

  <!-- Not implemented by EclipseLink yet
  <java-method name="testMapOfCustomer">
     <java-params>
         <java-param>
            <oxm:xml-element
 xmlns='http://www.eclipse.org/eclipselink/xsds/persistence/oxm'>
              <xml-map>
                        <key type='java.lang.String'/>
                        <value
 type='weblogic.wsee.databinding.internal.test.toplink.Customer'/>
              </xml-map>
            </oxm:xml-element>
         </java-param>
     </java-params>
  </java-method>
  -->

  <java-method name="testMapOfCustomerAdapters">
     <oxm:xml-element
 xmlns='http://www.eclipse.org/eclipselink/xsds/persistence/oxm'>
       <oxm:xml-java-type-adapter
 value='weblogic.wsee.databinding.internal.test.toplink.MapStringIntegerAdapter'/>
     </oxm:xml-element>
     <java-params>
```

```
        <java-param>
           <oxm:xml-element
 xmlns='http://www.eclipse.org/eclipselink/xsds/persistence/oxm'>
             <oxm:xml-java-type-adapter

value='weblogic.wsee.databinding.internal.test.toplink.MapStringCustomerAdapter'/>
           </oxm:xml-element>
        </java-param>
     </java-params>
  </java-method>

  <!-- Not implemented: Bare Multi-part -->
  <java-method name="test3Lists">
  <web-method exclude="true"/>
  </java-method>
  </java-methods>
</java-wsdl-mapping>
```

The EXM schema file for external mapping metadata for the data binding framework is available at the following URL:

http://www.oracle.com/technology/weblogic/weblogic-wsee-databinding/1.1/weblogic-wsee-databinding.xsd

The data defines the attributes of a particular Java web service endpoint. This schema defines three types of XML constructs:

- Constructs that are analogous to JAX-WS or JSR-181that override or define attributes on the service endpoint interface (SEI) and JAXB annotations for the value types utilized in the interfaces of the SEI.
- Additional mapping specifications not available using standard JAX-WS or JAXB annotations, primarily for use with the `java.util.Collections` API.
- References to external JAXB mapping metadata from a Toplink OXM file.

When a construct is the direct analog of a JAX-WS, JSR-181, or JAXB annotation, the comment in the schema contains a notation such as:

```
Corresponding Java annotation: javax.jws.WebParam.Mode
```

Part X

Using Oracle Business Activity Monitoring

This part describes Oracle Business Activity Monitoring.

This part contains the following chapters:

50

Integrating Oracle BAM with SOA Composite Applications

This chapter provides information about using the Oracle BAM Adapter in the SOA composite applications using Oracle JDeveloper.

This chapter contains the following topics:

50.1 Introduction to Integrating Oracle BAM with SOA Composite Applications

The Oracle BAM Adapter is a Java Connector Architecture (JCA)-compliant adapter that can be used from a Java EE client to send data and events to the Oracle BAM Server. The Oracle BAM Adapter supports the following operations on Oracle BAM data objects: inserts, updates, upserts, and deletes.

The Oracle BAM Adapter can perform these operations over Remote Method Invocation (RMI) calls (if they are deployed in the same farm), direct Java object invocations (if they are deployed in the same container), or over Simple Object Access Protocol (SOAP) (if there is a fire wall between them).

Oracle BAM Adapter is configured in Oracle WebLogic Server Administration Console to provide any of these connection pools. See Section 50.2, "Configuring Oracle BAM Adapter" for more information.

Some configuration is required to connect SOA composite applications to Oracle BAM. See Section 50.4, "Creating a Design Time Connection to an Oracle BAM Server" for more information.

Oracle BAM Adapter can be used with various features in SOA composite applications by which you can send data to an Oracle BAM Server:

- The Oracle BAM Adapter transfers data from BPEL process monitors to automatically generated Oracle BAM data objects. See Section 50.3, "Using Oracle BAM Monitor Express With BPEL Processes" for more information.
- The Oracle BAM Adapter can be used as a reference binding component in a SOA composite application. For example, Oracle Mediator can send data to Oracle BAM using the Oracle BAM Adapter. See Section 50.5, "Using Oracle BAM Adapter in a SOA Composite Application" for more information.
- The Oracle BAM Adapter can also be used as a partner link in a Business Process Execution Language (BPEL) process to send data to Oracle BAM as a step in the process. See Section 50.6, "Using Oracle BAM Adapter in a BPEL Process" for more information.
- Oracle BAM sensor actions (which use Oracle BAM Adapter) can be included within a BPEL process to publish event-based data to the Oracle BAM data objects. See Section 50.7, "Integrating BPEL Sensors Using Oracle BAM Sensor Action" for more information.

JMS sensor actions on BPEL sensors can feed data to Oracle BAM, and circumvent Oracle BAM Adapter. See Section 50.8, "Integrating SOA Applications and Oracle BAM Using Enterprise Message Resources" for more information.

JMS sensor actions at the SOA composite application level can feed data to Oracle BAM. See Chapter 47, "Defining Composite Sensors" for more information.

50.2 Configuring Oracle BAM Adapter

The Oracle BAM Adapter Java Naming and Directory Interface (JNDI) connection pools must be configured when you use the Oracle BAM adapter to connect with the Oracle BAM Server at runtime. For information about configuration see "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

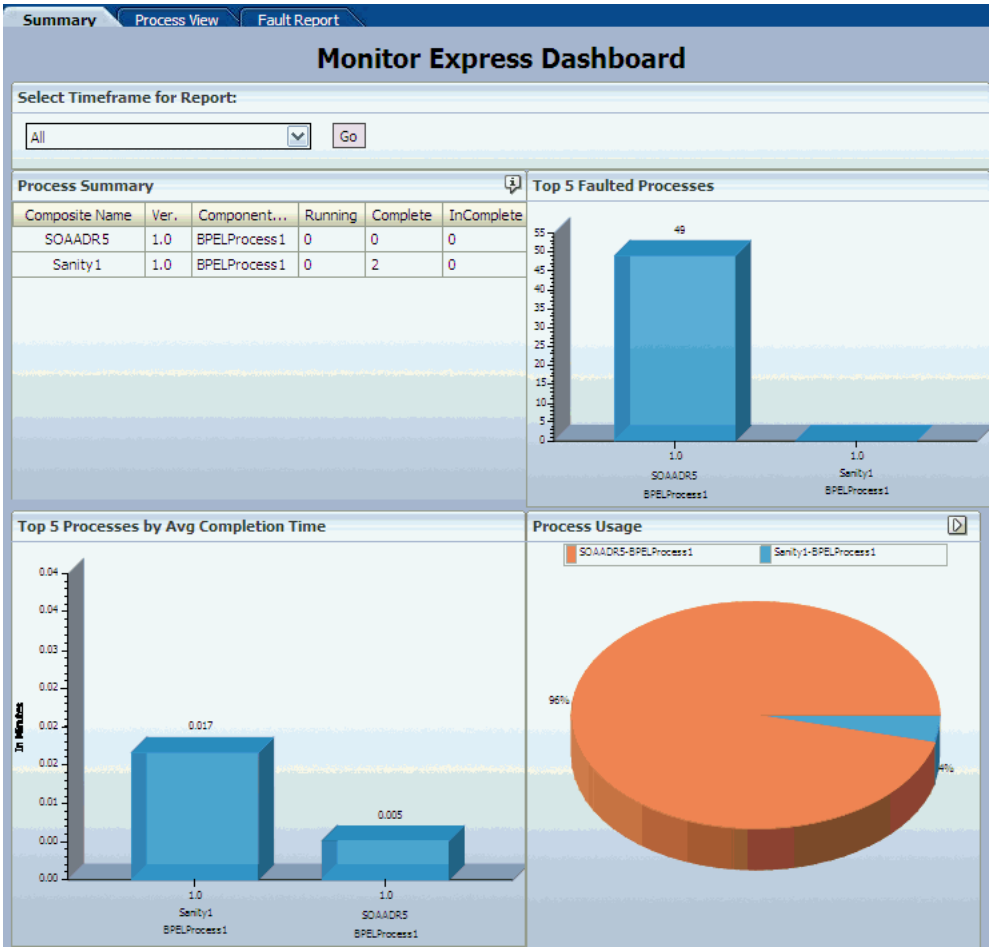
Make note of the JDNI names that you configure in the Oracle BAM Adapter properties, so that you can use them in the Oracle BAM Adapter wizard, Monitor Express configuration, and the Oracle BAM sensor action configuration in Oracle JDeveloper.

When using an RMI connection between a SOA composite application and Oracle BAM Server, that is, when they are deployed in different domains, trusted domain configuration must be done in Oracle WebLogic Server Administrative Console. See "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

50.3 Using Oracle BAM Monitor Express With BPEL Processes

The Monitor Express offering from Oracle BAM provides high level instrumentation of BPEL processes, automatically handling Oracle BAM data object deployment and population.

Figure 50–1 Oracle BAM Monitor Express Dashboard



Activity Monitors and Monitoring Objects are used to capture BPEL process metrics, which are sent to Oracle BAM Server, and then used for analysis and graphic display. All of the connection, design, and deployment configuration is accomplished in Oracle JDeveloper.

Monitor Express ships with sample dashboards to demonstrate solutions you can build on top of the automatically deployed data objects. You can also build custom dashboards on the data objects generated by Monitor Express using Oracle BAM Active Studio or with Oracle BAM data controls in an ADF application.

Using the BPEL Designer Monitor view in Oracle JDeveloper, you can create the following types of monitors on a BPEL process:

- *Activity Monitors* capture running time data for BPEL process activities, scopes, and human tasks. Activity Monitors can help identify bottlenecks in the BPEL process. See Section 50.3.2, "How to Configure Activity Monitors" for more information.
- *Counter* monitoring objects capture the date and time when a particular BPEL activity event is encountered within the BPEL process. Counters may be useful for reporting the number of times a particular activity is executed over a period of time. See Section 50.3.4, "How to Configure Counters" for more information.
- *Interval* monitoring objects capture the amount of time for the process to go from one BPEL activity event to another. Interval monitoring objects can help identify

bottlenecks in the BPEL process. See Section 50.3.5, "How to Configure Intervals" for more information.

- *Business Indicator* monitoring objects capture a snapshot of BPEL variables or expressions at a specified activity event in the BPEL process. See Section 50.3.6, "How to Configure Business Indicators" for more information.

When the SOA composite application is deployed, the Oracle BAM data objects corresponding to the BPEL process monitors are created or updated automatically.

This section contains the following topics:

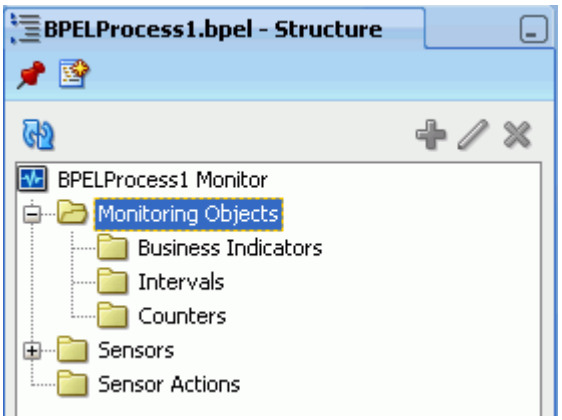
Related Documentation

- Chapter 52, "Defining and Managing Oracle BAM Data Objects"
- *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring*

50.3.1 How to Access BPEL Designer Monitor View

To access BPEL Designer Monitor view, select **Monitor** in the BPEL Designer toolbar.

In Monitor view, the structure pane displays the **Monitoring Objects** folder. You can expand the folder to expose the **Business Indicators**, **Intervals**, and **Counters** folders.

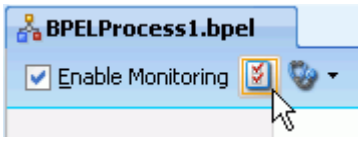


50.3.2 How to Configure Activity Monitors

Configure and enable Activity Monitors to capture data on start and end times for the BPEL process including the individual BPEL activities, scopes, and human tasks.

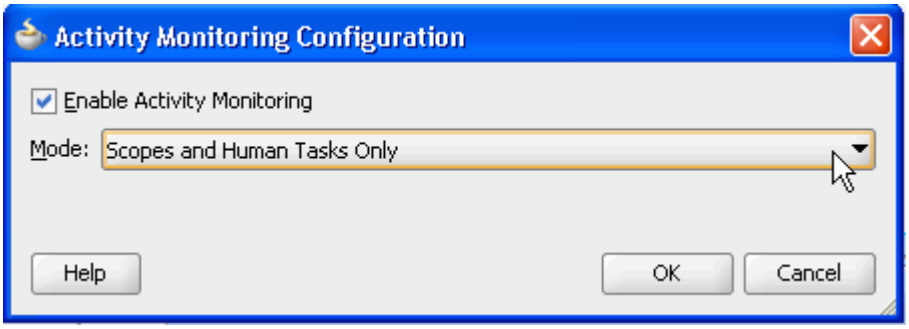
To configure Activity Monitors:

1. In the Monitor view of a BPEL process, click **Activity Monitoring Configuration** in the BPEL Designer tool bar.



> **Note:** The global **Enable Monitoring** flag overrides the local setting.

2. In the Activity Monitoring Configuration dialog, select **Enable Activity Monitoring**, and choose the **Mode** to configure the level of monitoring.



- The **All Activities** option captures start and end time data for every activity in the BPEL process, including individual activities, scopes, and human tasks. An activity starts when the activation event for the activity is begun, and it ends when the completion event is finished.
- The **Scopes and Human Tasks Only** option captures start and end time data for every scope and human task defined in the BPEL process. A scope starts when the first activity activation event within the scope is begun, and it ends when the final activity completion event within the scope is finished. A human task activity starts when the activation event for the human task

activity is begun, and it ends when the completion event in the human task activity is finished.

- The **Human Tasks Only** option captures start and end time data for every human task activity defined in the BPEL process.
- The **BPEL Process Only** option captures start and end time data for the BPEL process.

You can disable Activity Monitors by deselecting the **Enable Activity Monitoring** checkbox.

3. Click **OK**.

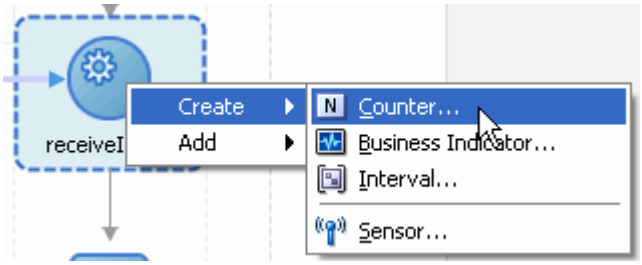
   If Activity Monitors are enabled, data is sent to Oracle BAM data object at runtime. See Section 50.3.10, "What You Need To Know About Monitor Express Data Objects" for more information about Oracle BAM data objects for monitoring objects.

50.3.3 How To Create BPEL Process Monitoring Objects

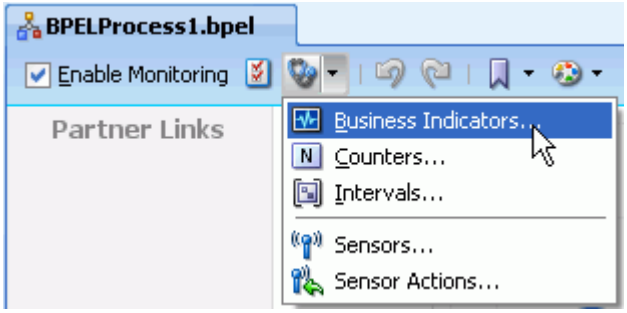
Use the BPEL Designer Monitor view in Oracle JDeveloper to create BPEL process monitoring objects.

To create a BPEL process monitoring object:

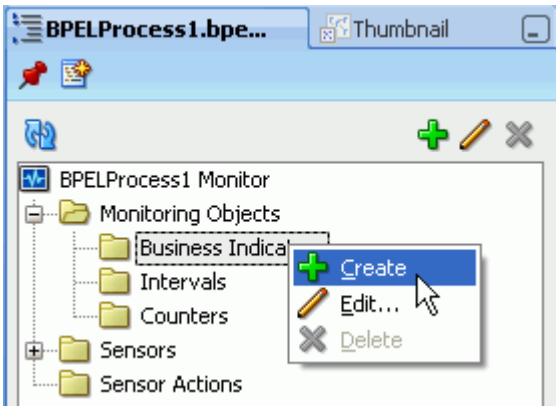
1. While in the Monitor view, open a context menu on an activity in the BPEL process diagram, select **Create**, and choose a monitoring object type from the list.



Alternatively, you can use the **Monitoring Objects** menu, located at the top left corner of the BPEL Designer window, to create monitoring objects.



As another alternative, you can open a context menu for each **Monitoring Objects** type folder in the Structure pane to create a monitoring object.

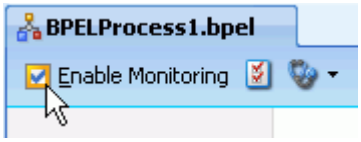


BPEL process configurable monitoring objects are available in three types: Counters, Intervals, and Business Indicators. See the following topics for more information.

- Section 50.3.4, "How to Configure Counters"
- Section 50.3.5, "How to Configure Intervals"
- Section 50.3.6, "How to Configure Business Indicators"

2. To enable the BPEL process monitoring objects at run time, verify that the **Enable Monitoring** checkbox, located at the top left corner of the BPEL Designer Monitor view, is selected.

Figure 50–2 Enable Monitoring Checkbox



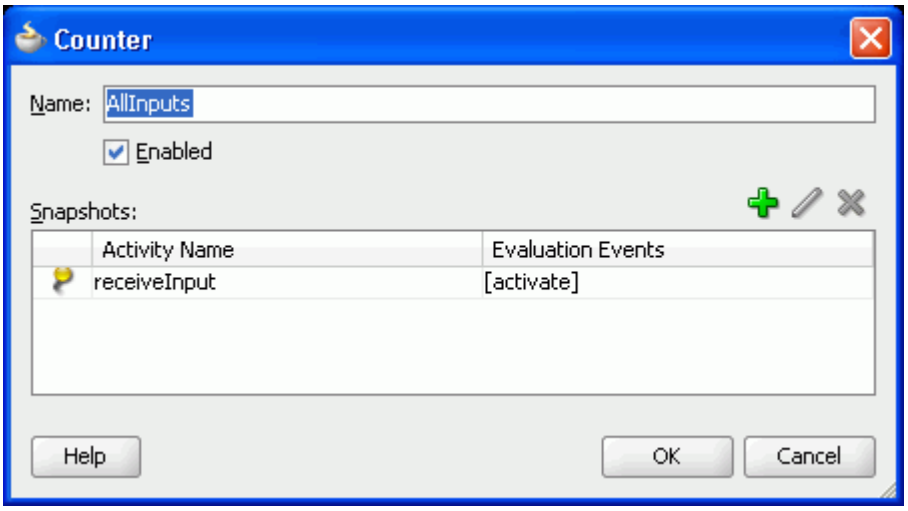
When checked, the **Enable Monitoring** option in BPEL Designer enables all of the monitors and sensors in all BPEL processes in the current SOA composite application. It overrides any monitoring object-level enable flags.

When the **Enable Monitoring** option is not checked, a property called `enableProcessSensors` is added to `composite.xml` with the value `false`. That property disables all monitors and sensors in all BPEL processes in the current SOA composite application.

50.3.4 How to Configure Counters

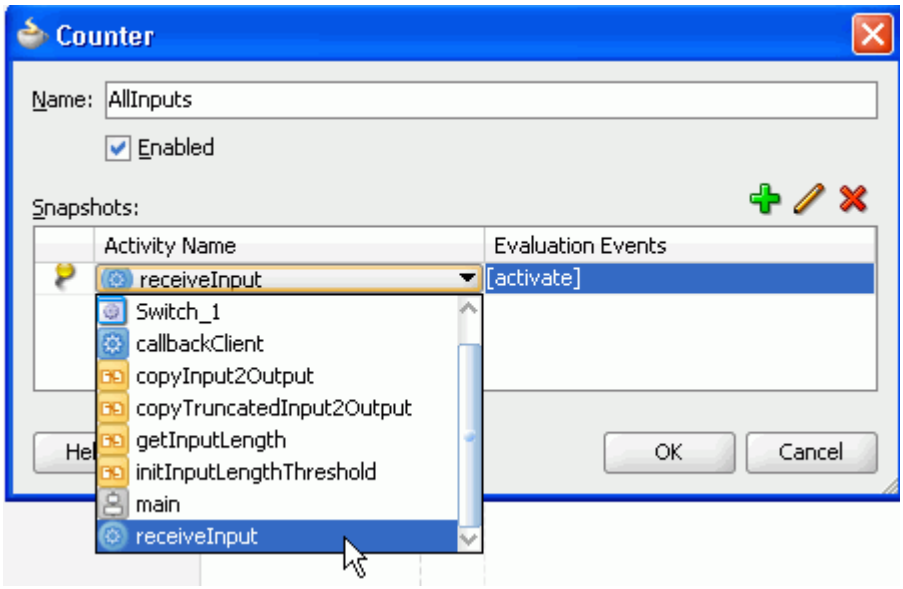
Every time the BPEL process passes a snapshot of a Counter (which is attached to an activity in the BPEL process diagram), data is sent to Oracle BAM. The Counter indicates how often a BPEL activity is encountered, and creates a new record in an Oracle BAM data object with time data.

Use the Counter dialog to configure a Counter monitoring object.

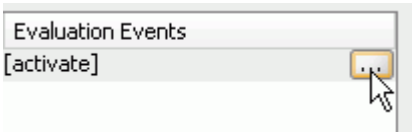


The **Enabled** checkbox enables or disables this particular monitoring object. If it is not enabled, the Counter is not evaluated during the BPEL process, therefore no data is sent to Oracle BAM.

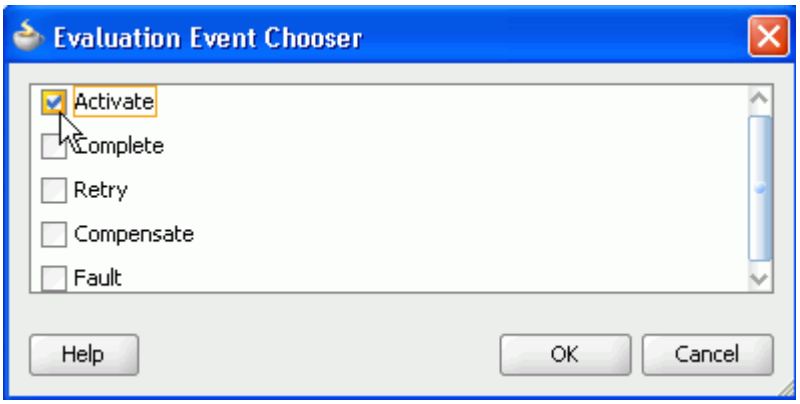
To attach a snapshot of a Counter to a BPEL activity, click the **Add** icon in the Counter dialog. Then select an activity from the list.



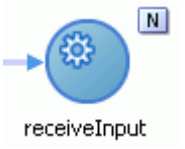
Next, choose an evaluation event (an event within the activity), by clicking the browsing icon.



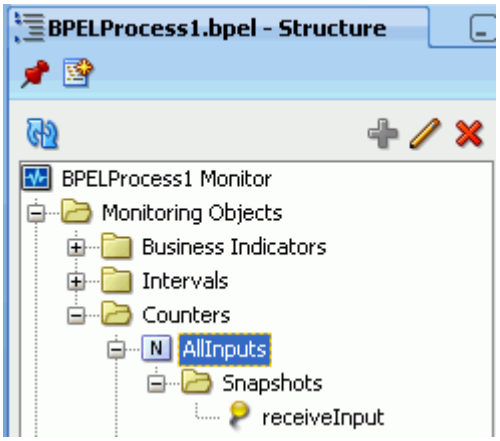
The Evaluation Event Chooser opens to let you select one or more evaluation events.



When the Counter snapshot configuration is complete, it is displayed as an N icon next to activity in the BPEL process diagram.



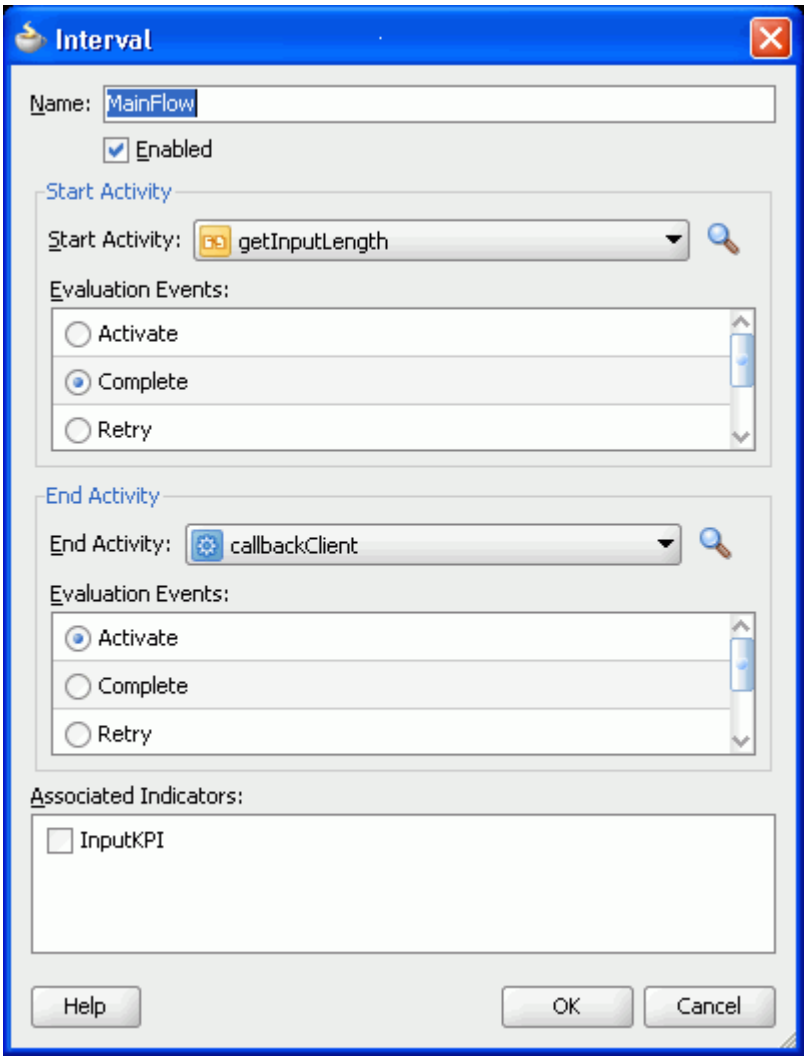
The Counter and its snapshot are represented in the structure pane.



50.3.5 How to Configure Intervals

An Interval monitoring object captures the amount of time to go from one activity to another in the BPEL process. The start and end times are captured and sent to an Oracle BAM data object.

Use the Interval dialog to configure an Interval monitoring object.



The **Enabled** checkbox enables or disables this particular monitoring object. If it is not enabled, the Interval is not evaluated during the BPEL process, therefore no data is sent to Oracle BAM.

The **Start Activity** defines the beginning of the Interval. Select a **Start Activity** from the list, and a single selection in the **Evaluation Events** list.

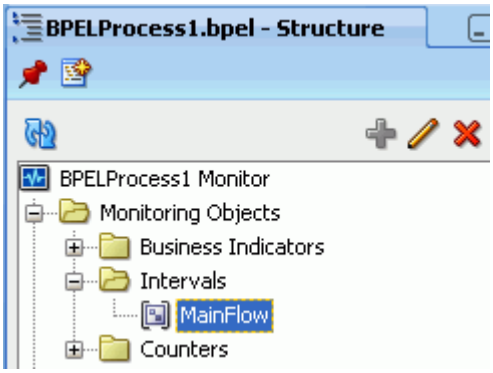
End Activity defines the end of the Interval. Select an **End Activity** from the list, and a single selection in the **Evaluation Events** list.

You can select **Associated Indicators** if a Business Indicator has been previously defined in the BPEL process. Selecting an associated indicator automatically provides two snapshots on the selected Business Indicator. This captures the Business Indicator metrics at the start and at the end of the Interval.

Note: If you plan to include an associated indicator snapshot in the Interval, it is not recommended to use the `main` or `receiveInput` activities at the Activate evaluation event as the start or end points, because the variables in the XPath expression might not yet be populated.

BPEL activities of type `receive`, typically named `receiveInput`, allow the process to wait for a matching message to arrive. The arriving message is copied to a variable specified in the definition of the activity. The copy operation occurs between the `activate` and `complete` evaluation events, and not before or on `activate`. Therefore, caution must be taken when defining monitoring object snapshots on BPEL activities of type `receive`, especially if the `activate` evaluation event is chosen.

The Interval is represented in the structure pane.



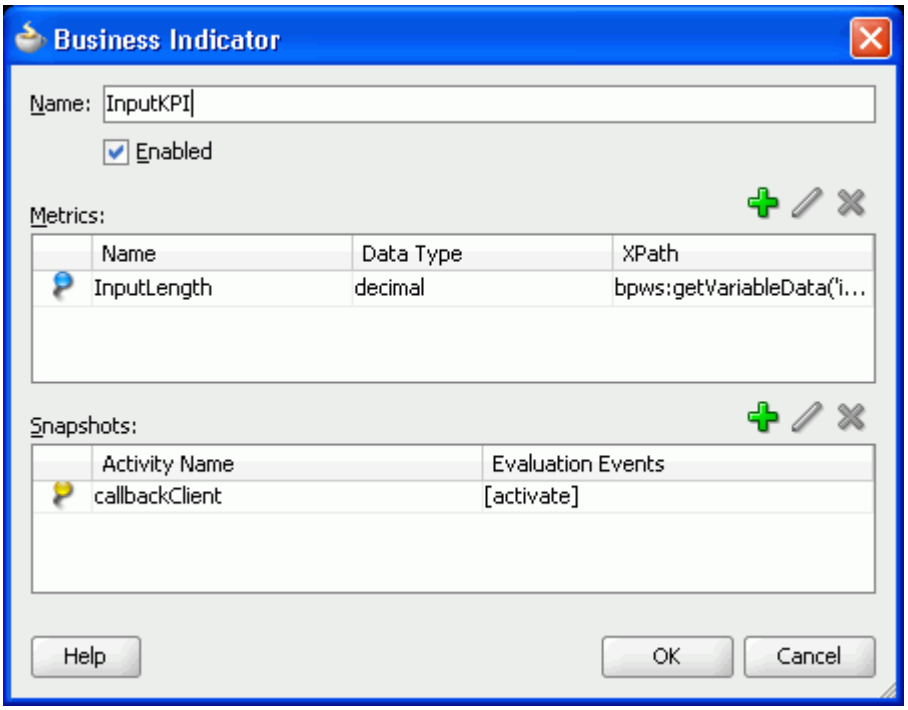
On execution, the Interval start and end times are sent to Oracle BAM as a new record in a data object. See Section 50.3.10, "What You Need To Know About Monitor Express Data Objects" for information about the Oracle BAM data objects.

An empty Interval, one in which the start and end activities and evaluation events are the same, is valid, and it can label Business Indicator snapshots. The Interval can uniquely identify multiple snapshots for a single Business Indicator. Instead of configuring snapshots in the Business Indicator dialog, you can create an empty Interval for each snapshot you want to create for a Business Indicator, and select the Business Indicator's indicator reference in each Interval.

50.3.6 How to Configure Business Indicators

A Business Indicator monitoring object captures a snapshot of BPEL variables, specified by the metrics in the Business Indicator, or evaluates expressions, when the events specified in the Business Indicator are encountered in the BPEL process.

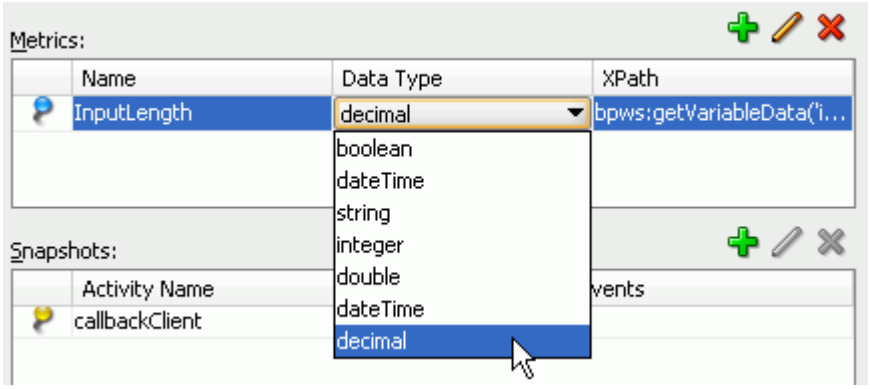
Use the Business Indicator dialog to configure a Business Indicator monitoring object.



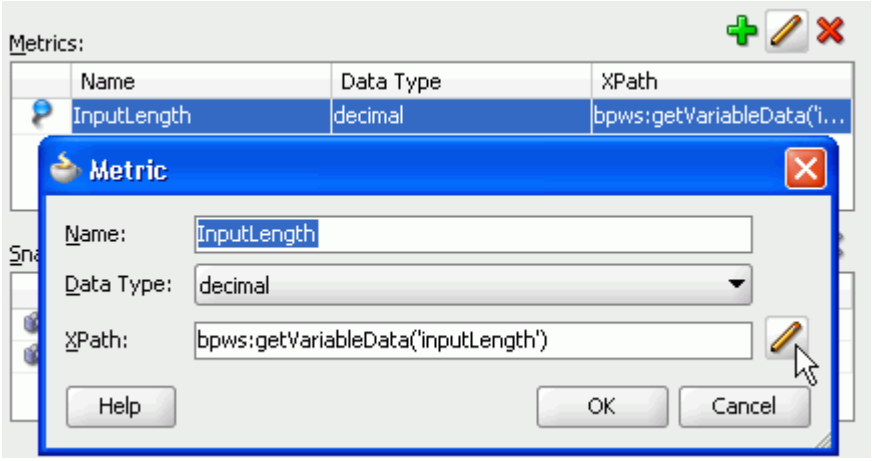
The **Enabled** checkbox enables or disables this particular monitoring object. If it is not enabled, the configured expression in the Business Indicator is not evaluated during the BPEL process, therefore no data is sent to Oracle BAM.

Metrics are defined to evaluate an expression or variable when the events specified in the Business Indicator are encountered in the BPEL process.

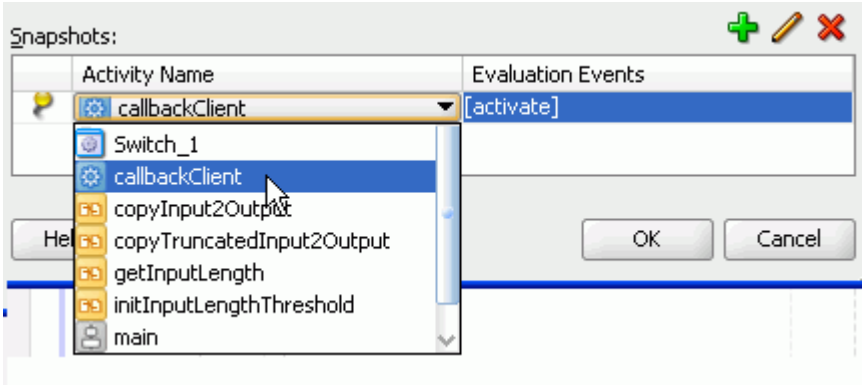
Click the green plus icon to configure a metric. Metrics have a name, data type, and XPath expression.



You can enter an expression directly in the **XPath** field, or click **Edit** to open the Metric configuration dialog, and click **Edit** to use the **Expression Builder**.



Snapshots associate the Business Indicator with activities in the BPEL process. The snapshot tells the BPEL process at what point to evaluate the Business Indicator metrics. To create a snapshot, click the green plus icon.



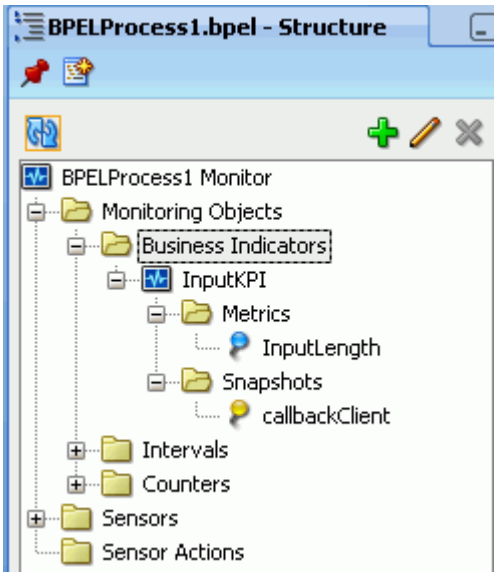
> **Note:** You can use empty Interval monitoring objects to uniquely identify snapshots of a particular Business Indicator. See Section 50.3.5, "How to Configure Intervals" for more information.

Evaluation Events indicate at what point during the activity to evaluate the Business Indicator metrics. Select a Snapshot in the table and click **Edit** to select one or more evaluation events. You can pick multiple evaluation events within the BPEL activity on which to evaluate the metric.

> **Note:** Configuring a snapshot on the `main` or `receiveInput` activities at the Activate evaluation event is not recommended because the variables in the XPath expression might not yet be populated.

When the configuration is saved, a Business Indicator icon is displayed in the top right corner of the associated activity in the BPEL process diagram.

The Business Indicator is also represented in the structure pane with its metrics and snapshots.



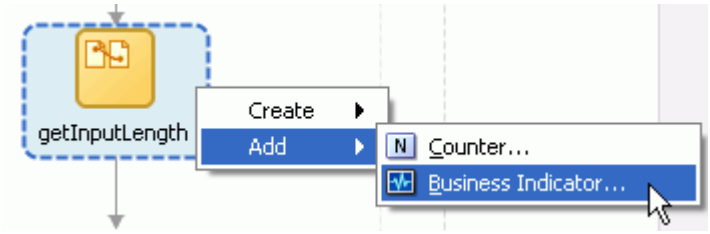
On execution, when a Business Indicator is encountered in the BPWL process, the metrics are evaluated and results sent to Oracle BAM as new records in a data object. See Section 50.3.10, "What You Need To Know About Monitor Express Data Objects" for information about the Oracle BAM data objects.

50.3.7 How to Add Existing Monitoring Objects to Activities

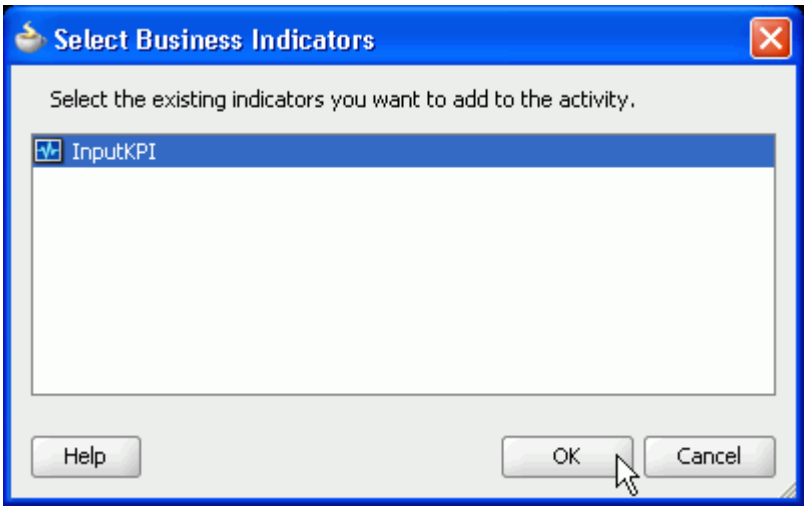
You can add previously created Counters and Business Indicators to activities in the BPEL process with a shortcut menu provided in the BPEL Designer Monitor view. This creates a new snapshot in the selected Counter or Business Indicator.

To add a monitor to an activity:

1. Right-click the activity to which you want to add the monitor, and select **Add**.



2. Select **Counter** or **Business Indicator**.
3. Select one or more monitoring objects in the dialog and click **OK**. Press Shift-click to select multiple monitoring objects.



4. An icon appears within the activity boundary.

50.3.8 How To Configure BPEL Process Monitors for Deployment

When any BPEL process in the current SOA composite application contains monitoring objects, during the deployment of that composite, Oracle BAM data objects are created in Oracle BAM Server in the location specified in the `monitor.config` file.

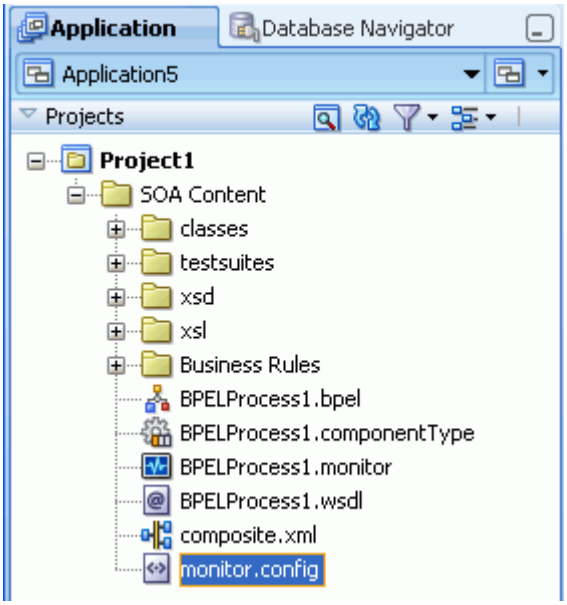
Note: The `monitor.config` file is created on demand. If there are no monitors configured in the BPEL process, there is no `monitor.config` file.

The `monitor.config` file does not appear automatically in Oracle JDeveloper Application Navigator when the first monitor object is created. The user must save all of the files in the project (using **Save All**), and then refresh the Application Navigator.

Deployment is incremental, meaning that existing data objects are not deleted, and columns are added to data objects when required by the monitoring object configuration. See Section 50.3.10, "What You Need To Know About Monitor Express Data Objects" for details about the data objects.

To configure deployment properties:

In the Application Navigator project folder, open the `monitor.config` file for editing.



The `monitor.config` file defines deployment and runtime properties needed to connect with Oracle BAM Server to create and populate the data objects.

> **Caution:** Do *not* edit the *`BPELProcess`*`.monitor` file. It is an internal file, and it must not be edited manually. It stores the metadata for all of the BPEL process monitors in the specific BPEL process.

The default `monitor.config` file is shown in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<MonitorConfig>
  <Connection>
    <BAM dataObjectsFolder="/Samples/Monitor Express/"
         adapterConnectionFactoryJNDI="eis/bam/rmi" batch="true"
         deploymentProtocol="http">
    </BAM>
  </Connection>
  <Deployment ignoreErrors="true"/>
</MonitorConfig>
```

The properties are described in Table 50–1.

Define only one Connection block per BPEL project.

Table 50–1 Monitor Configuration Properties

Property	Default	Description
`dataObjectsFolder`	/Samples/Monitor Express/	Path to the location of the data objects for the monitors configured in all of the BPEL process for the SOA composite application. If the directory does not exist, it is created during deployment. The path is relative to the root data object folder in Oracle BAM Server. Note that there is only one data objects folder per SOA composite application. The application can contain many BPEL processes. All of the data objects associated with all of the BPEL processes in the application are created in this location.
`adapterConnectionFactoryJNDI`	eis/bam/rmi	Oracle BAM Adapter connection pool configured in Oracle WebLogic Server Administration Console. Oracle BAM Adapter must be configured before deployment and runtime. When using the RMI protocol, as when Oracle SOA Server and Oracle BAM Server are deployed in separate domains, you must also configure trusted domain credentials for both Oracle SOA Server and Oracle BAM Server domains. See Section 50.2, "Configuring Oracle BAM Adapter" and *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.
`batch`	true	Indicates that batching using Oracle BAM Adapter is enabled. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for information about batching configuration properties.
`deploymentProtocol`	http	The only valid value is `http`.

Table 50–1 (Cont.) Monitor Configuration Properties

Property	Default	Description
`ignoreErrors`	true	If Oracle BAM Server is unreachable or there are any problems with the deployment of the Oracle BAM data objects, and this property is set to `true`, deployment of the composite does not halt. If set to `false` and Oracle BAM Server is unavailable, the deployment fails. This property corresponds to the **Ignore BPEL Monitor deployment errors** checkbox in the deployment configuration wizard.

50.3.9 What You Need to Know About Using the Monitor Express Dashboard

Oracle BAM provides a sample dashboard that you can use to monitor your BPEL process out of the box.

The Monitor Express dashboard and data object samples allow users to enable Oracle BAM for your SOA composite applications in relatively few steps from within Oracle JDeveloper. The ready-to-use dashboards provide a single integrated view to track Key Performance Indicators (KPIs) in real-time and promote operational efficiency. The rich user experience for monitoring is delivered by BPEL Monitor instrumentation in Oracle JDeveloper.

The data objects are located in the `Samples/Monitors/` data object directory in Oracle BAM Architect, and the sample reports are located in the `Shared Reports/Samples/Monitor Express/` folder in Oracle BAM Active Viewer.

If the samples are not installed on your system, the installation script and instructions are located in the *`SOA_ORACLE_HOME`*`/bam/samples/bam/monitorexpress` directory.

50.3.10 What You Need To Know About Monitor Express Data Objects

Oracle BAM data objects are deployed automatically when a SOA composite application containing enabled BPEL process monitors is deployed. Preseeded sample data objects are present in the `Samples/Monitor Express/` directory.

You can use these data objects to construct Oracle BAM dashboards. See *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for information about creating dashboards in Oracle BAM Active Studio.

You can add columns and indexes to the data objects using Oracle BAM Architect. The custom columns and indexes you add in Oracle BAM Architect are preserved when a revised SOA composite application containing changes to BPEL process monitor configuration is deployed. See Chapter 52, "Defining and Managing Oracle BAM Data Objects" for information about adding columns and indexes.

If a data object already exists in the configured location at deployment time, it is used as is, or updated with the appropriate additional columns to accommodate messages from the BPEL process monitors.

Oracle BAM data objects cannot be changed if they are in use. If there are Oracle BAM dashboards open against BPEL process monitor data objects, and the data objects require changes upon deployment, the data object updates fail.

Note: Do not change the existing monitoring data object column names.

Oracle BAM Adapter Configuration

BPEL process monitors use Oracle BAM Adapter to convey messages to Oracle BAM Server. At deployment time, if Oracle BAM Server is unreachable, deployment fails. If Oracle BAM Server is unreachable at runtime, the retry behavior is determined by the Oracle BAM Adapter configuration. See Section 50.2, "Configuring Oracle BAM Adapter" and "Configuring Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

See the following sections for a detailed description of the data objects and troubleshooting information.

- Section 50.3.10.1, "Understanding the COMPONENT Data Object"
- Section 50.3.10.2, "Understanding the COUNTER Data Object"
- Section 50.3.10.3, "Understanding the INTERVAL Data Object"
- Section 50.3.10.4, "Understanding Business Indicator Data Objects"
- Section 50.3.10.5, "Troubleshooting"

50.3.10.1 Understanding the COMPONENT Data Object

The COMPONENT data object is the main dimension table. It compiles information about how long a BPEL process instance takes to run, and if it has failed at least once.

This data object is always populated when at least one monitoring object is configured or if you have activity monitoring enabled.

Table 50–2 COMPONENT Data Object Fields

Column Name	Description
COMPOSITE_INSTANCE_ID	SCA composite instance ID number.
COMPONENT_INSTANCE_ID	SCA component instance ID number. For BPEL it is the BPEL instance ID number.
DOMAIN_NAME	The partition name.
COMPOSITE_NAME	The name of the SOA composite application.
COMPOSITE_REVISION	The revision number of the SOA composite application.
COMPOSITE_LABEL	SOA composite application internal label. This label is created every time you deploy even if you override the revision ID.
COMPONENT_TYPE	The component type (BPEL, for a BPEL process, for example).
COMPONENT_NAME	The component display name (The name of a BPEL process, for example).

Table 50–2 (Cont.) COMPONENT Data Object Fields

Column Name	Description
COMPONENT_START_TIME	The date and time that the component started running.
COMPONENT_END_TIME	The date and time that the component stopped running.
COMPONENT_FAULT_FLAG	Indicates whether the component has faulted at least once. 1=faulted, 0=no fault.
FAULT_NAME	Name of the last fault that occurred.
COMPONENT_RUNNING_FLAG	Indicates whether the component is currently running. 1=the component is running, 0=the component is not running.
COMPONENT_RUNNING_TIME_IN_SEC	The calculated length of time between COMPONENT_START_TIME and COMPONENT_END_TIME in seconds.
COMPONENT_RUNNING_TIME_IN_MIN	The calculated length of time between COMPONENT_START_TIME and COMPONENT_END_TIME in minutes.
COMPONENT_COMPLETED_NO_FAULT_FLAG	Indicates whether the component completed with no faults. 1=completed with no fault, 0=either did not complete yet, or did complete with fault.
COMPONENT_INCOMPLETE_FLAG	Indicates that the component has not completed, and has faulted at least once. 1=has not completed, and has faulted at least once, 0=otherwise.
ECID	Allows you to create reports that aggregate data over multiple BPEL process instances executed by a single SOA request.

50.3.10.2 Understanding the COUNTER Data Object

The COUNTER data object contains data captured by all of the Counter monitoring objects encountered in the BPEL processes.

Table 50–3 COUNTER Data Object Fields

Column Name	Description
COMPOSITE_INSTANCE_ID	SCA composite instance ID number.
COMPONENT_INSTANCE_ID	SCA component instance ID number. For BPEL it is the BPEL instance ID number.
DOMAIN_NAME	Lookup to DOMAIN_NAME field in COMPONENT data object.
COMPOSITE_NAME	Lookup to COMPOSITE_NAME field in COMPONENT data object.
COMPOSITE_REVISION	Lookup to COMPOSITE_REVISION field in COMPONENT data object.
COMPOSITE_LABEL	Lookup to COMPOSITE_LABEL field in COMPONENT data object.
COMPONENT_TYPE	Lookup to COMPONENT_TYPE field in COMPONENT data object.

Table 50–3 (Cont.) COUNTER Data Object Fields

Column Name	Description
COMPONENT_NAME	Lookup to COMPONENT_NAME field in COMPONENT data object.
COMPONENT_START_TIME	Lookup to COMPONENT_START_TIME field in COMPONENT data object.
COMPONENT_END_TIME	Lookup to COMPONENT_END_TIME field in COMPONENT data object.
COMPONENT_FAULT_FLAG	Lookup to COMPONENT_FAULT_FLAG field in COMPONENT data object.
FAULT_NAME	Lookup to FAULT_NAME field in COMPONENT data object.
COUNTER_NAME	The name of the Counter monitoring object.
SUBCOMPONENT_ID	An internal value that is used as a key field.
SUBCOMPONENT_TYPE	Type of the sub-component (sequence indicates a BPEL sequence activity, for example) where the Counter data was captured. The human task type is used for Human Task activities.
SUBCOMPONENT_NAME	Name of the sub-component (receiveInput, for example) where the Counter data was captured. In BPEL it is the name of the activity.
EVALUATION_EVENT	The event within the life cycle of the BPEL activity (activate, for example) at which the data is captured.
SNAPSHOT_TIME	Date and time when the Counter data was captured.

50.3.10.3 Understanding the INTERVAL Data Object

The INTERVAL data object contains data captured by all of the Interval monitoring objects and Activity Monitors configured in the BPEL processes.

Table 50–4 INTERVAL Data Object Fields

Column Name	Description
COMPOSITE_INSTANCE_ID	SCA composite instance ID number.
COMPONENT_INSTANCE_ID	SCA component instance ID number. For BPEL it is the BPEL instance ID number.
DOMAIN_NAME	Lookup to DOMAIN_NAME field in COMPONENT data object.
COMPOSITE_NAME	Lookup to COMPOSITE_NAME field in COMPONENT data object.
COMPOSITE_REVISION	Lookup to COMPOSITE_REVISION field in COMPONENT data object.
COMPOSITE_LABEL	Lookup to COMPOSITE_LABEL field in COMPONENT data object.
COMPONENT_TYPE	Lookup to COMPONENT_TYPE field in COMPONENT data object.
COMPONENT_NAME	Lookup to COMPONENT_NAME field in COMPONENT data object.
COMPONENT_START_TIME	Lookup to COMPONENT_START_TIME field in COMPONENT data object.

Table 50–4 (Cont.) INTERVAL Data Object Fields

Column Name	Description
COMPONENT_END_TIME	Lookup to COMPONENT_END_TIME field in COMPONENT data object.
COMPONENT_FAULT_FLAG	Lookup to COMPONENT_FAULT_FLAG field in COMPONENT data object.
FAULT_NAME	Lookup to FAULT_NAME field in COMPONENT data object.
INTERVAL_NAME	Display name of the Interval monitoring object, or the name of the activity, human task, or scope being monitored by Activity Monitors.
INTERVAL_TYPE	Indicates the type of BPEL process monitor where the data was captured. CUSTOM indicates an Interval monitoring object configured with custom start and end times. Interval monitoring objects are described in Section 50.3.5, "How to Configure Intervals." SUBCOMPONENT indicates an Activity Monitor. Activity Monitors are described in Section 50.3.2, "How to Configure Activity Monitors."
INTERVAL_START_TIME	Date and time recorded when the Interval or Activity Monitor start activity was encountered.
INTERVAL_END_TIME	Date and time recorded when the Interval or Activity Monitor end activity was encountered.
START_SUBCOMPONENT_ID	An internal value that is used as a key field.
START_SUBCOMPONENT_TYPE	The type of the BPEL process activity being monitored by an interval. The human task type is used for Human Task activities.
START_SUBCOMPONENT_NAME	The display name of the process activity being monitored by an interval.
START_EVALUATION_EVENT	The event within the life cycle of the BPEL activity (activate, for example) at which the data is captured.
END_SUBCOMPONENT_ID	An internal value that is used as a key field.
END_SUBCOMPONENT_TYPE	The type of the BPEL process activity being monitored by an interval. The human task type is used for Human Task activities.
END_SUBCOMPONENT_NAME	The display name of the process activity being monitored by an interval.
END_EVALUATION_EVENT	The event within the life cycle of the BPEL activity (activate, for example) at which the data is captured.
SUBCOMPONENT_CREATOR	For future use.
INTERVAL_RUNNING_FLAG	Indicates if the Interval or Activity Monitor end activity is running. 1 indicates that the end activity has not been encountered. 0 indicates otherwise.
INTERVAL_RUNNING_TIME_IN_SEC	The length of time between the INTERVAL_START_TIME and INTERVAL_END_TIME in seconds.
INTERVAL_RUNNING_TIME_IN_MIN	The length of time between the INTERVAL_START_TIME and INTERVAL_END_TIME in minutes.

50.3.10.4 Understanding Business Indicator Data Objects

The data objects containing data captured by all of the Business Indicator metrics configured in a BPEL process are named BI_*Partition_Name_Composite_Name_BPELPROCESS_Name*.

A separate data object is created for each BPEL process in the SOA composite application that contains Business Indicator monitoring objects.

If a Business Indicator is referenced by an Interval monitoring object, some of the data related to the Interval (`INTERVAL_NAME`, `INTERVAL_START_FLAG`, and `INTERVAL_END_FLAG`) is captured in the Business Indicator data object.

> **Note:** If one of the metrics fails at the time of evaluation (snapshot) the data is not sent to Oracle BAM; however, the remaining metrics configured in the Business Indicator are evaluated at the snapshot. If the failed Business Indicator metric is encountered at another snapshot, the BPEL engine attempts to evaluate it.

Table 50–5 Business Indicator Data Object Fields

Column Name	Description
`COMPOSITE_INSTANCE_ID`	SCA composite instance ID number.
`COMPONENT_INSTANCE_ID`	SCA component instance ID number. For BPEL it is the BPEL instance ID number.
`DOMAIN_NAME`	Lookup to `DOMAIN_NAME` field in COMPONENT data object.
`COMPOSITE_NAME`	Lookup to `COMPOSITE_NAME` field in COMPONENT data object.
`COMPOSITE_REVISION`	Lookup to `COMPOSITE_REVISION` field in COMPONENT data object.
`COMPOSITE_LABEL`	Lookup to `COMPOSITE_LABEL` field in COMPONENT data object.
`COMPONENT_TYPE`	Lookup to `COMPONENT_TYPE` field in COMPONENT data object.
`COMPONENT_NAME`	Lookup to `COMPONENT_NAME` field in COMPONENT data object.
`COMPONENT_START_TIME`	Lookup to `COMPONENT_START_TIME` field in COMPONENT data object.
`COMPONENT_END_TIME`	Lookup to `COMPONENT_END_TIME` field in COMPONENT data object.
`COMPONENT_FAULT_FLAG`	Lookup to `COMPONENT_FAULT_FLAG` field in COMPONENT data object.
`FAULT_NAME`	Lookup to `FAULT_NAME` field in COMPONENT data object.
`BI_NAME`	Name of the Business Indicator.
`SNAPSHOT_TIME`	Date and time recorded when the Business Indicator data was captured.
`SUBCOMPONENT_ID`	An internal value that is used as a key field.

Table 50–5 (Cont.) Business Indicator Data Object Fields

Column Name	Description
SUBCOMPONENT_TYPE	Type of the subcomponent (`invoke` indicates a BPEL invoke activity, for example) where the Business Indicator data was captured. The `human task` type is used for Human Task activities.
SUBCOMPONENT_NAME	Name of the subcomponent (`callbackClient`, for example) where the Business Indicator data was captured
EVALUATION_EVENT	The event within the life cycle of the BPEL activity (activate, for example) at which the data is captured.
INTERVAL_NAME	The name of the Business Indicator-instrumented Interval monitoring object that lead to the Business Indicator data capture. The field is null if the data was captured within an Activity Monitor.
INTERVAL_START_FLAG	Indicates whether the data was captured at the Interval start activity. 1=yes, NULL=otherwise. The field is null if the data was captured within an Activity Monitor.
INTERVAL_END_FLAG	Indicates whether the data was captured at the Interval end activity. 1=yes, NULL=otherwise. The field is null if the data was captured within an Activity Monitor.
LATEST	Indicates (with value "Y") the latest snapshot of a Business Indicator record for a particular composite/component instance (based on COMPOSITE_INSTANCE_ID and COMPONENT_INSTANCE_ID). Allows the creation of dashboards that filter Business Indicator records so only the latest is used (a Business Indicator can have many snapshots in the same process, but LATEST indicates the most recent at any point in time).
METRIC_*NAME*	Contains the result of the XPath expression evaluated in the *NAME* metric. Each METRIC_*NAME* field is the data type configured in the metric. The *NAME* portion of these column names is the display name of the metrics configured in the Business Indicators. There are as many METRIC_*NAME* fields as there are metrics configured in the BPEL process. Metric names must be unique within a BPEL process to avoid name collisions in this data object.

50.3.10.5 Troubleshooting

This section contains Monitor Express troubleshooting information.

50.3.10.5.1 Controlling Oracle BAM Data Object Size

In Oracle BAM Server data objects, older data can be purged with an alert rule, so that the data object does not grow too large.

See Chapter 57, "Creating Oracle BAM Alerts" for general information alerts, and see Section F.3.8, "Delete rows from a Data Object" for information about configuring the delete action.

50.3.10.5.2 Using the Logs

Monitor Express runtime logs messages using the `oracle.soa.bpel.engine.sensor` logger. For more information, see Configuring Log Files in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

50.4 Creating a Design Time Connection to an Oracle BAM Server

You must create a connection to an Oracle BAM Server to browse the available data objects and construct transformations while you are designing your applications in Oracle JDeveloper.

> **Note:** Oracle BAM Server connections should be created in Application Resources, directly, or by copying an existing connection from the Resource Catalog.

50.4.1 How to Create a Connection to an Oracle BAM Server

You create a connection to an Oracle BAM Server to browse data objects available on that server.

To create a connection to an Oracle BAM Server:

1. From the **File** main menu in Oracle JDeveloper, select **New**.

   The New Gallery dialog box opens.

2. From the **General** category, choose **Connections**.
3. From the **Items** list, select **BAM Connection**, and click **OK**.

   The BAM Connection wizard opens.

4. Ensure that **Application Resources** is selected.
5. Provide a name for the connection.
6. Click **Next**.
7. Enter the connection information about the Oracle BAM Server host described in Table 50–6.

Table 50–6 Oracle BAM Server Connection Information

Field	Description
BAM Web Host	Enter the name of the host on which the Oracle BAM Report Server and web applications are installed. In most cases, the Oracle BAM web applications host and Oracle BAM Server host are the same.
BAM Server Host	Enter the name of the host on which the Oracle BAM Server is installed.
User Name	Enter the Oracle BAM Server user name.
Password	Enter the password of the user name.

Table 50–6 (Cont.) Oracle BAM Server Connection Information

Field	Description
HTTP Port	Enter the port number or accept the default value of **9001**. This is the HTTP port for the Oracle BAM web applications host.
JNDI Port	Enter the port number or accept the default value of **9001**. The JNDI port is for the Oracle BAM report cache, which is part of the Oracle BAM Server.
Use HTTPS	Select this checkbox to use secure HTTP (HTTPS) to connect to the Oracle BAM Server during design time. Otherwise, HTTP is used.

8. Click **Next**.
9. Test the connection by clicking **Test Connection**. If the connection was successful, the following message appears:

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

3 of 3 tests successful.
```

10. Click **Finish**.

50.5 Using Oracle BAM Adapter in a SOA Composite Application

The Oracle BAM Adapter is used as a reference that enables the SOA composite application to send data to an Oracle BAM Server external to the SOA composite application.

50.5.1 How to Use Oracle BAM Adapter in a SOA Composite Application

You can add Oracle BAM Adapter references that enable the SOA composite application to send data to Oracle BAM Servers external to the SOA composite application.

To add an Oracle BAM Adapter reference:

1. In the Component Palette, select **SOA**.
2. Drag the **BAM Adapter** to the right swim lane.

   This launches the Adapter Configuration wizard.
3. In the Service Name page, provide a **Service Name** and an optional **Description**.
4. In the Data Object Operation and Keys page,
   a. Select a **Data Object** using the BAM Data Object Chooser dialog box.

      When you click Browse the Data Object Chooser dialog box opens allowing you to browse the available Oracle BAM Server connections in the **BAM Data Object Explorer** tree. Select a data object and click **OK**.
   b. Choose an **Operation** from the list.

      Insert adds a row to the data object.

Upsert inserts new data into an existing row in a data object if the row exists. If the row does not exist a new row is created. You must select a key from the **Available** column to upsert rows in a data object.

Delete removes a row from the data object. You must select a key from the **Available** column to delete rows in a data object.

Update inserts new data into an existing row in a data object. You must select a key from the **Available** column to update rows in a data object.

c. Provide an appropriate display name in the **Operation Name** field for this operation in your SOA composite application.

d. To select **Enable Batching** select the checkbox.

The data cached in memory by the Oracle BAM Adapter of the Oracle BPEL Process Manager runtime is flushed (sent) to Oracle BAM Server periodically. The Oracle BAM component may decide to send data before a batch timeout if the cache has some data objects between automatically defined lower and upper limit values.

Batching properties are configured in BAMCommonConfig.xml. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

5. In the JNDI Name page, specify the **JNDI Name** for the Oracle BAM Server connection.

The JNDI name is configured in the Oracle WebLogic Server Administration Console. See "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

6. Click **Finish**.

50.6 Using Oracle BAM Adapter in a BPEL Process

The Oracle BAM Adapter is used as a partner link in a BPEL process to send data to Oracle BAM as a step in the process.

For more information, see Section 4.3, "Introduction to Partner Links."

50.6.1 How to Use Oracle BAM Adapter in a BPEL Process

You can add the Oracle BAM Adapter to a BPEL process to send data to Oracle BAM as a step in the process. The Oracle BAM Adapter is used as a partner link and connected to an activity in the BPEL process.

To add an Oracle BAM partner link:

1. In the SOA Composite Editor in Oracle JDeveloper, double-click the BPEL process icon to open it in the BPEL Process Designer.
2. In the Component Palette, expand the BPEL Services panel.
3. Drag and drop the Oracle BAM Adapter into the Partner Links swim lane on the right side of the BPEL Process Designer.
4. In the Adapter Configuration wizard, enter a display name in the Service Name field and click Next.

When the wizard completes, a Web Services Description Language (WSDL) file by this name appears in the Application Navigator for the BPEL process or Oracle

Mediator message flow. This file includes the adapter configuration settings you specify with this wizard.

5. In the Data Object Operation and Keys page,
   a. Select a **Data Object** using the BAM Data Object Chooser dialog box.

      When you click Browse the Data Object Chooser dialog box opens allowing you to browse the available Oracle BAM Server connections in the **BAM Data Object Explorer** tree. Select a data object and click **OK**.

   b. Choose an **Operation** from the list.

      Insert adds a row to the data object.

      Upsert inserts new data into an existing row in a data object if the row exists. If the row does not exist a new row is created.

      Delete removes a row from the data object.

      Update inserts new data into an existing row in a data object.

   c. Provide an appropriate display name in the **Operation Name** field for this operation in your SOA composite application.

   d. To select **Enable Batching** select the checkbox.

      The data cached in memory by the Oracle BAM Adapter of the Oracle BPEL Process Manager runtime is flushed (sent) to Oracle BAM Server periodically. The Oracle BAM component may decide to send data before a batch timeout if the cache has some data objects between automatically defined lower and upper limit values.

      Batching properties are configured in BAMCommonConfig.xml. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

6. In the JNDI Name page, specify the **JNDI Name** for the Oracle BAM Server connection.

   The JNDI name is configured in the Oracle WebLogic Server Administration Console. See "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

7. Click **Finish**.

8. Create a new Process Variable in the BPEL process of type Message Type, and browse the Type Chooser dialog box to select the WDSL for the data object you want to write to on the Oracle BAM Server.

   For more information about using the Oracle BPEL Process Manager see Chapter 4, "Getting Started with Oracle BPEL Process Manager."

9. In the BPEL Process add an activity that you can use to map the source data to the new variable you created.

10. In the BPEL Process add an Invoke activity to send data to the Oracle BAM Adapter partner link you created. Add the variable you just created as the Input Variable.

11. Save all of the project files.

50.7 Integrating BPEL Sensors Using Oracle BAM Sensor Action

You can create sensor actions in Oracle BPEL Process Manager to publish sensor data into existing data objects on an Oracle BAM Server. When you create the sensor action, you can select an Oracle BPEL Process Manager variable sensor or activity sensor to get the data from and the data object in Oracle BAM Server in which you want to publish the sensor data.

The Oracle BAM Adapter supports batching of operations, but behavior with batching is different from behavior without batching. As the Oracle BAM Adapter is applied to BPEL sensor actions, the Oracle BAM sensor action is not part of the BPEL transaction. When batching is enabled, BPEL does not wait for an Oracle BAM operation to complete. It is an asynchronous call.

When batching is disabled, BPEL waits for the Oracle BAM operation to complete before proceeding with the BPEL process, but it does not roll back or stop when there is an exception from Oracle BAM. The Oracle BAM sensor action logs messages to the same sensor action logger as BPEL. See "Configuring Oracle BAM Batching Properties" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for information about batching behavior.

These instructions assume you have installed and configured Oracle BAM.

Notes: Connection factory configuration must be completed before using Oracle BAM sensor actions. Also, if the Oracle BAM Adapter is using credentials rather than a plain text user name and password, in order for the Oracle BAM Adapter (including Oracle BAM sensor actions used in BPEL) to connect to the Oracle BAM Server the credentials must also be established and mapped. See "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

50.7.1 How to Create a Sensor

Before you can create an Oracle BAM sensor action, you must first create a sensor in the BPEL process. You must create a sensor before creating a Oracle BAM sensor action.

- Variable sensor

  Restrictions: A Variable sensor's variable must be defined in a standalone XSD. This variable must not be defined inline in the WSDL file. If the variable has message parts, then there must be only one message part.

- An Activity sensor containing exactly one sensor variable

  Restrictions: Because you map the sensor data to a single Oracle BAM Server data object, the Activity sensor must contain only one variable. All of the Variable sensor restrictions also apply.

Note: Any sensor that does not conform to these rules are be filtered from the Oracle BAM sensor action configuration dialog box. Also, if a sensor is created conforming to the restrictions, but the variable is deleted (rendering the sensor invalid), it does not appear in Oracle BAM sensor action configuration dialog box.

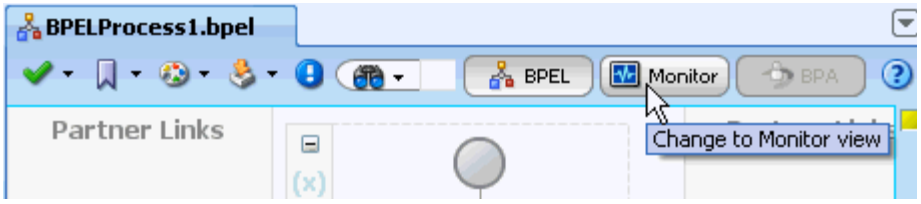
For more information about creating sensors, see Section 17.2, "Configuring Sensors and Sensor Actions in Oracle JDeveloper."

50.7.2 How to Create an Oracle BAM Sensor Action

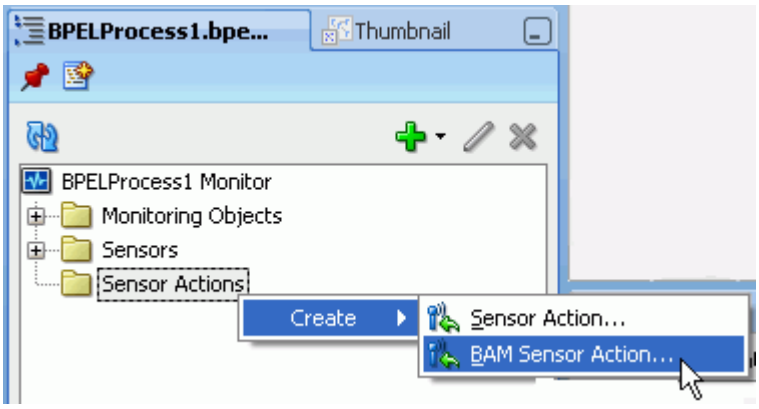
When you create the Oracle BAM sensor action, you select the BPEL variable sensor or activity sensor from which to get data, and you select the data object in Oracle BAM Server to which you want to publish the sensor data.

To create an Oracle BAM sensor action:

1. Go to your BPEL process in Oracle JDeveloper.
2. Select Monitor from the BPEL Designer menu in the upper right corner.



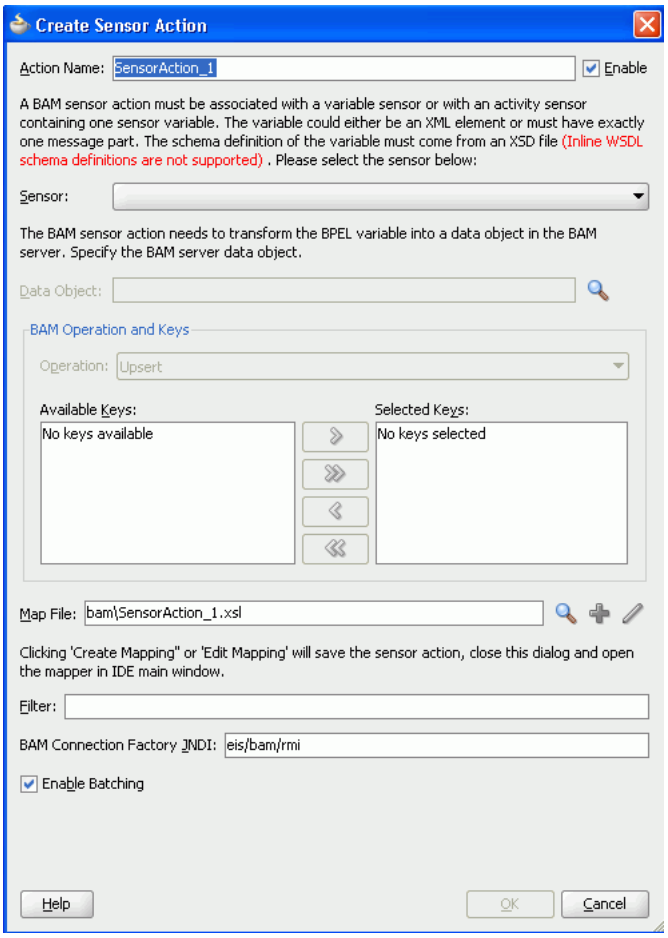
3. In the Structure window, select and right-click **Sensor Actions**.



If the Structure window is not open, select **View** > **Structure Window** to open it.

4. Select **Create** > **BAM Sensor Action**.

The Create Sensor Action dialog box appears.



5. Enter the details described in Table 50–7:

Table 50–7 Create Sensor Action Dialog Box Fields and Values

Field	Description
Action Name	Enter a unique and recognizable name for the sensor action.
Enable	Select this option to enable the sensor action. When disabled no sensor action data is sent to Oracle BAM. Sensors can be also be disabled using Oracle Enterprise Manager Fusion Middleware Control.
Sensor	Select a BPEL sensor to monitor. This is the sensor that you created in Section 50.7.1, "How to Create a Sensor" for mapping sensor data to a data object in Oracle BAM Server.
Data Object	Click the **Browse** icon to open the BAM Data Object Chooser dialog box to select the data object in Oracle BAM Server in which you want to publish the sensor data. If you have not created a connection to Oracle BAM Server to select data objects, click the icon in the upper right corner of the BAM Data Object Chooser dialog box.
Operation	Select to **Delete**, **Update**, **Insert**, or **Upsert** a row in the Oracle BAM Server database. Upsert first attempts to update a row if it exists. If the row does not exit, it is inserted.

Table 50–7 (Cont.) Create Sensor Action Dialog Box Fields and Values

Field	Description
Available Keys/Selected Keys	If you selected the **Delete**, **Update**, or **Upsert** operation, you must also select a column name in the Oracle BAM Server database to use as a key to determine the row with which this sensor object corresponds. A key can be a single column or a composite key consisting of multiple columns. Select a key and click the **>** button. To select all, click the **>>** button.
Map File	Provide a file name to create a mapping between the sensor data (selected in the **Sensor** list) and the Oracle BAM Server data object (selected in the **Data Object** list). You can also invoke a mapper dialog box by clicking the **Create Mapping** icon (second icon) or **Edit Mapping** icon (third icon).
Filter	Enter an XPath expression to filter sensor action data that is sent to Oracle BAM. At runtime the XPath expression entered in the field is evaluated, and it must return true for the sensor action to be fired. Enter filter logic as a boolean expression. A filter enables you to monitor sensor data within a specific range. For example, you may want to monitor loan requests in which the loan amount is greater that $100,000. In this case, you can enter an expression such as the following: `boolean(/s:actionData/s:payload/s:variableData/s:data/autoloan:loanAmount > 100000)` See Figure 17–9, "Creating a Sensor Action with a Filter" for an example.
BAM Connection Factory JNDI	Specify the JNDI name for the Oracle BAM Server connection factory. The JNDI name is configured in the Oracle WebLogic Server Administration Console. See "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.
Enable Batching	The data accumulated by the Oracle BAM component of the Oracle BPEL Process Manager runtime is flushed (sent) to Oracle BAM Server periodically. The Oracle BAM component may decide to send data before a batch timeout if the queue has some data objects between automatically defined lower and upper limit values. If batching is enabled, performance is dramatically improved, but there is no transaction guarantee. The BPEL process continues to run without waiting for the data to get to the Oracle BAM Server. If batching is not enabled, the BPEL process waits until the Oracle BAM Server confirms that the record operation was completed; however, if there is a failure, the exception from Oracle BAM Server is logged and the BPEL process continues. BPEL does not roll back the operation or stop when there is an exception from Oracle BAM. See "Configuring Oracle BAM Batching Properties" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for information about batching behavior.

WARNING: If you restart Oracle BPEL Server, any messages currently being batched are lost. Ensure that all messages have completed batching before restarting Oracle BPEL Server.

Notes: After you click the **Create Mapping** or **Edit Mapping**, or the **OK** button on the Create Sensor Action dialog box, you must explicitly save the BPEL file.

6. Click **OK** to close the Create Sensor Action dialog box.

   You must complete the XSLT mapping the sensor action XML schema to the Oracle BAM data object schema.

50.8 Integrating SOA Applications and Oracle BAM Using Enterprise Message Resources

You can use BPEL JMS sensor actions to send data to Oracle BAM from a SOA composite application by way of a JMS topic or queue, using Oracle BAM Enterprise Message Sources.

You can also use the generic JMS adapter at the SOA composite or BPEL level, and Enterprise Message Sources can read that data into Oracle BAM.

XSL must be used to transform the payload from the BPEL JMS sensor action. You can use the advanced XML processing option in Oracle BAM Enterprise Message Sources, including using XSL, to get to any attribute or node in the XML.

See the following documentation for more information:

- Chapter 53, "Creating Oracle BAM Enterprise Message Sources"
- Section 17.2, "Configuring Sensors and Sensor Actions in Oracle JDeveloper."
- "Oracle JCA Adapter for JMS" in *Oracle Fusion Middleware User's Guide for Technology Adapters*

51

Using Oracle BAM Data Control

Oracle BAM data control is a binding component in the Oracle ADF Model with support for Active Data Services. This chapter provides information about creating and using Oracle BAM data control.

For more comprehensive information about using Oracle ADF Model data binding and Active Data Services, refer to *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

This chapter contains the following topics:

51.1 Introduction to Oracle BAM Data Control

Oracle BAM data control allows ADF developers to build applications with a dynamic user interface that changes based on real-time business events. Oracle BAM data control is used to bind data from Oracle BAM data objects to databound UI components in an ADF page.

Oracle BAM data control abstracts a query on Oracle BAM data objects using standard metadata interfaces to describe the Oracle BAM data collections. Using JDeveloper, you can view that information as icons which you can drag and drop onto a page. Using those icons, you can create databound UI components (for JSF JSP pages) by dragging and dropping them from the Data Controls panel onto the visual editor for a page. JDeveloper automatically creates the metadata that describes the bindings from the page to the Oracle BAM data objects. At runtime, the ADF Model layer reads the metadata information from appropriate XML files for both the data controls and bindings and implements the connection between your user interface and Oracle BAM data objects. Note that Oracle BAM data control is read-only.

For general information about Oracle ADF data controls, and information about ADS (active data services), see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

51.2 Creating Projects That Can Use Oracle BAM Data Controls

Oracle BAM data control must to be hosted by a valid ADF web application. Also, a limited set of ADF Faces components support active data, therefore a limited set of ADF Faces components can make use of the main functionality of an Oracle BAM data control. Refer to Oracle JDeveloper ADF documentation for information about creating ADF web applications, including a list of components that support active data.

Note that an Oracle BAM data control can still be used by view components that do not support active data.

Oracle BAM data control requires that the project contain the ADF Faces and ADF Page Flow technologies. The Fusion Web Application (ADF) template in JDeveloper contains these technologies.

51.3 Creating Oracle BAM Server Connections

You must create a connection to Oracle BAM to browse the available data objects in JDeveloper. This connection information is automatically used during deployment and runtime. See Section 50.4, "Creating a Design Time Connection to an Oracle BAM Server" for details on creating the connection.

Note: Oracle BAM data control only uses connections that appear in the Application Resources panel. It does not find connections in the Resource Palette. Oracle JDeveloper facilitates copying connections from Resource Palette to the Application Resources panel of your application.

Note: To create an Oracle BAM data control against an SSL-enabled Oracle BAM Server Oracle JDeveloper must be started with the `-J-Djavax.net.ssl.trustStore` option, which should point to the location of the key store. The connection to Oracle BAM Server cannot be created without this option.

Example:

```
C:\jdevrc1\jdeveloper\jdev\bin>jdev
-J-Djavax.net.ssl.trustStore=C:\jdevrc1\wlserver_
10.3\server\lib\DemoTrust.jks
```

51.3.1 How to Modify Oracle BAM Data Control Connections to Oracle BAM Servers

Each Oracle BAM data control has an associated Oracle BAM connection. When a connection has changed name or has been removed from the application resources, you get an error when you attempt to use any data controls that are associated with the connection. You can do one of the following to resolve the lost connection:

- Create a new Oracle BAM connection with the same name as the connection that is referred to by the data control. See Section 51.3, "Creating Oracle BAM Server Connections" for more information.
- Update the current project's DataControls.dcx file with the name of a new or existing Oracle BAM connection. See Section 51.3.1.1, "How to Associate a BAM Data Control with a New Oracle BAM Connection" for more information.

51.3.1.1 How to Associate a BAM Data Control with a New Oracle BAM Connection

To change the Oracle BAM connection associated with a particular data control you must edit the DataControls.dcx file in the current project. Change the `connection` attribute of the `BAMDataControl` element with the name of the desired Oracle BAM connection.

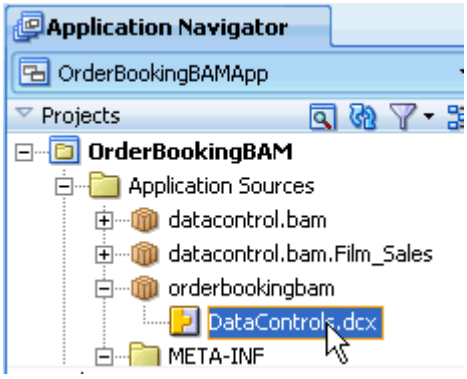
To modify the Oracle BAM connection in an Oracle BAM data control:

1. Optionally, create a new Oracle BAM connection in the application.

   If you do not have a BAM connection in the Application Resources to use for this data control, create a new one. See Section 51.3, "Creating Oracle BAM Server Connections" for more information.

2. Locate the DataControls.dcx file in the project, and open it for editing.

   The DataControls.dcx file is located in the Application Sources directory under the node named for the project.



   Each project in a ADF application has a DataControl.dcx file associated with it. Each DataControls.dcx file may have one or more data control definitions. If the current project does not contain the definition for the data control you want to modify, look through the other projects in the current application to locate it.

3. In the Source view, locate the appropriate data control definition, and locate the `BAMDataControl` element within it.

   In the source view find the AdapterDataControl block with the id that matches the display name of your data control.

```
<AdapterDataControl id="Film_Sales"
                    FactoryClass="oracle.tip.tools.ide.bam.dc.dt.adapter.BAMDa
                    ImplDef="oracle.tip.tools.ide.bam.dc.dt.adapter.Definition
                    SupportsTransactions="false"
                    SupportsSortCollection="false" SupportsResetState="false"
                    SupportsRangesize="false" SupportsFindMode="false"
                    SupportsUpdates="false"
                    Definition="datacontrol.bam.Film_Sales"
                    BeanClass="datacontrol.bam.Film_Sales"
                    xmlns="http://xmlns.oracle.com/adfm/datacontrol">
  <Source>
    <BAMDataControl xmlns="http://xmlns.oracle.com/bam/datacontrol"
                    connection="BAMServerConnection1">
```

4. Change the `connection` attribute to the name of the new Oracle BAM connection.

5. Save and close the DataControls.dcx file.

51.4 Exposing Oracle BAM with Oracle ADF Data Controls

Once you have created your Oracle BAM data objects and established a connection to an Oracle BAM server from JDeveloper, you can use JDeveloper to create data controls that provide the information needed to declaratively bind UI components to those data objects. Data controls consist of many XML metadata files that define the capabilities of the service that the bindings can work with at runtime.

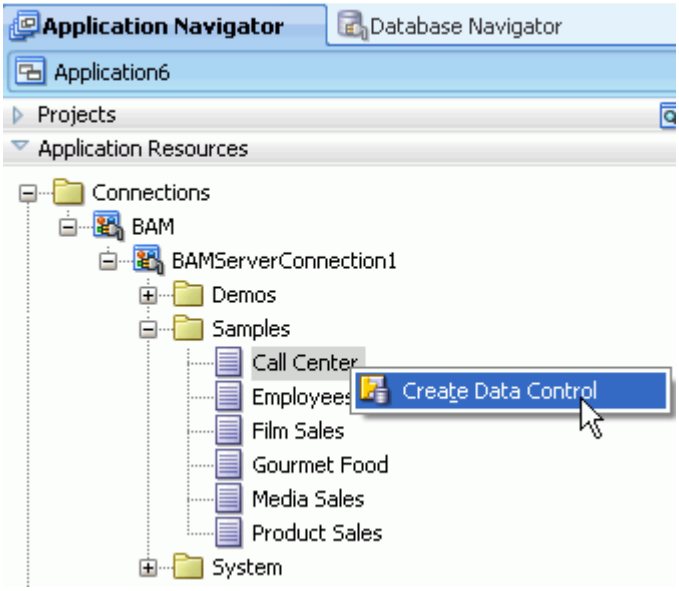
See Chapter 52, "Defining and Managing Oracle BAM Data Objects" for information about creating Oracle BAM data objects. For information about creating a connection to your Oracle BAM instance, see Section 51.3, "Creating Oracle BAM Server Connections."

51.4.1 How to Create Oracle BAM Data Controls

You create Oracle BAM data controls from within the Application Navigator of JDeveloper.

To create a data control:

1. In the Application Navigator, Application Resources panel, expand the data object folders in the Oracle BAM Server connection.
2. Right-click the Oracle BAM data object for which you want to create a data control, and select **Create Data Control** from the context menu.



3. Complete the BAM Data Control wizard to create the data control query.

   See Section 51.5, "Creating Oracle BAM Data Control Queries" for more information.

51.4.2 What Happens in Your Project When You Create an Oracle BAM Data Control

When you create a data control based on an Oracle BAM data object, the data control contains a representation of a query on all of the selected fields that is constructed based on the groupings, aggregates, filters, parameters, and additional calculated fields that you configure using the BAM Data Control wizard in JDeveloper.

For the data control to work directly with the service and the bindings, JDeveloper creates the following metadata XML files:

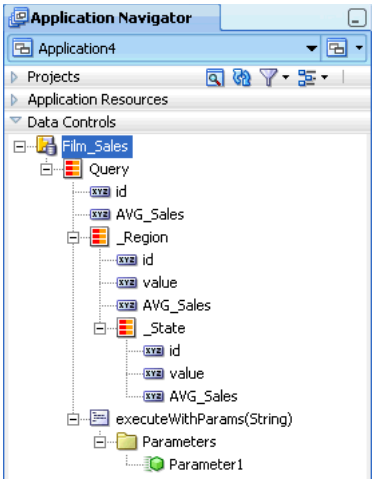
- Data control definition file (`DataControls.dcx`)
- Structure definition files for every structured object that this service exposes
- Design time XML files

JDeveloper also adds the icons to the Data Controls panel that you can use to create data bound UI components.

51.4.2.1 How an Oracle BAM Data Control Appears in the Data Controls Panel

The Data Controls panel lists all the data controls that have been created for the application's business services and exposes all the queries that are available for binding to UI components. The panel is a direct representation of the structure of the data to be returned by the data control. By editing the data control, you can change the elements displayed in the panel.

Figure 51–1 Data Controls Panel in Oracle JDeveloper



51.5 Creating Oracle BAM Data Control Queries

You can design a databound user interface by dragging an item from the Data Controls panel and dropping it on a page as a specific UI component. When you use Oracle BAM data controls to create a UI component, JDeveloper automatically creates the various code and objects needed to bind the component to the data control you selected.

You can create an Oracle BAM data control query using the Oracle BAM Data Control wizard. The wizard lets you choose between creating a flat query or a group query.

The following sections explain how to use each page in the wizard to create your query:

- Section 51.5.1, "How to Choose a Query Type"
- Section 51.5.2, "How to Create Parameters"
- Section 51.5.4, "How to Create Calculated Fields"
- Section 51.5.5, "How to Select, Organize, and Sort Fields"
- Section 51.5.6, "How to Create Filters"

- Section 51.5.7, "How to Select and Organize Groups"
- Section 51.5.8, "How to Create Aggregates"

51.5.1 How to Choose a Query Type

On the Name page of the Oracle BAM Data Control wizard, in addition to naming the data control and selecting the metadata XML files location, you can choose to create either a flat query or a group query.

In the **BAM Data Control Name** field, enter a display name for the data control.

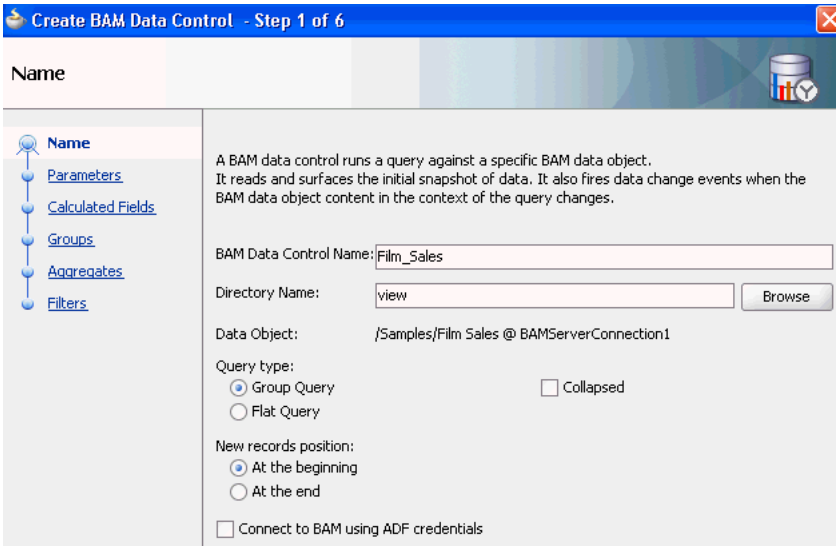
In the **Directory Name** field, enter the directory in which the data control metadata XML files are saved.

The **Data Object** path displays the location of the data object from which the query is built.

Select **Group Query** when you want to create groups and aggregates of data to display in trees or charts. The **Collapsed** checkbox, enabled only when **Group Query** is selected, makes the structure of the group query flat.

Select **Flat Query** when you want to show the data in a flat table or list.

In **New records position** select whether the new records are added to the beginning or end of the graph. For example, if you want new bars to appear on the right side of a bar graph, select **At the end**. If you want new rows inserted at the top of a list, select **At the beginning**.



Select the **Connect to BAM using ADF credentials** checkbox to connect to Oracle BAM Server at runtime using the credentials in the ADF application containing the Oracle BAM data control. This feature takes advantage of row-level security provided by Oracle BAM Server by using the ADF application user's identity to display only the data that the user is permitted to see.

To use this feature, both Oracle BAM Server and the ADF server must use the same credential store. When this feature is disabled (unchecked) the runtime connects to Oracle BAM Server using the credentials provided in the Oracle BAM connection, specified in Oracle JDeveloper or Oracle Enterprise Manager Fusion Middleware Control.

For more information about row-level security, see Section 52.6, "Creating Security Filters."

51.5.2 How to Create Parameters

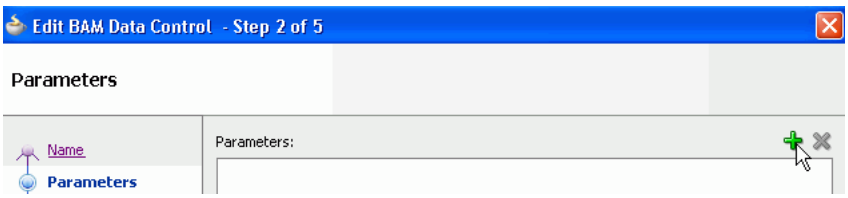
On the Parameters page of the Oracle BAM Data Control wizard you can create parameters that are used to pass values to filters on the Filters page of the wizard. For more information about creating filters see Section 51.5.6, "How to Create Filters."

For information about passing values to parameters, see Section 51.5.3, "How to Pass Values to Parameters."

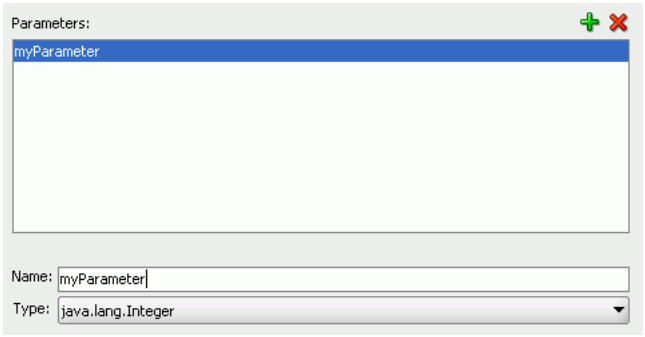
To create parameters:

1. Click **Add** to add a parameter.

   Click the **Add** icon above and to the right of the **Parameters** box.



2. To rename the parameter enter the text in the **Name** field.



3. Select the data type from the **Type** list.

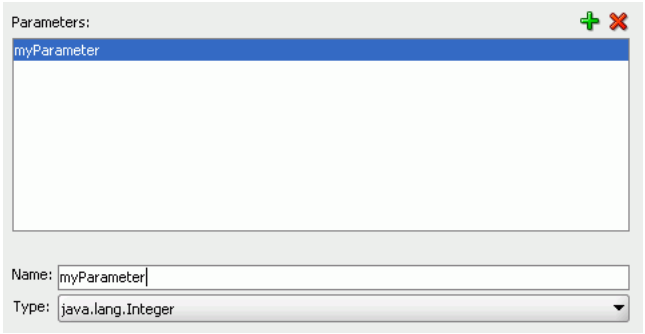


Table 51–1 Oracle BAM and Java Type Mapping

Java Type	Oracle BAM Type
java.lang.Integer	Integer
java.lang.String	String

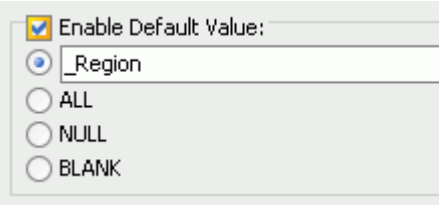
Table 51–1 (Cont.) Oracle BAM and Java Type Mapping

Java Type	Oracle BAM Type
java.util.Date	DateTime, Timestamp
java.lang.Boolean	Boolean
java.lang.BigDecimal	Decimal
jave.lang.Double	Float
	Field*

*The **Field** parameter type is used in charts for specifying groups at runtime. This parameter type allows the user to choose which field in the data object to group by. See the following topics for more information:

- Section 51.5.7, "How to Select and Organize Groups"
- Section 51.5.3, "How to Pass Values to Parameters"

4. To provide a default value for the parameter when loading the data control query, select **Enable Default Value** and choose a default value.

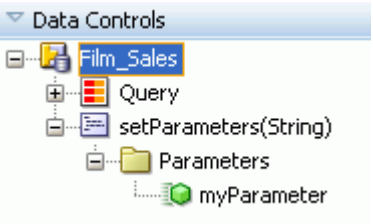


To enter a default value for the parameter, select one of the available defaults, or select the first option and enter a value in the field.

- **ALL** returns rows containing all values.
- **NULL** returns rows containing null values.
- **BLANK** returns rows containing blank string values.

51.5.3 How to Pass Values to Parameters

The operation `setParameters` appears in the Oracle BAM data control structure every time an Oracle BAM data control query is created with parameters.



To pass parameters to an Oracle BAM data control, the `setParameters` operation must be called in Oracle BAM data control before the query is executed.

One of the many ways that can be done is by using an ADF parameter form. For more information, see *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

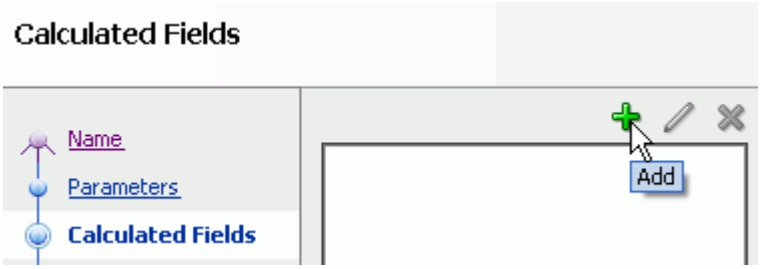
51.5.4 How to Create Calculated Fields

Calculated fields allow you to create new columns based on data derived from existing fields without updating the physical data object. Use the Oracle BAM Data Control wizard Calculated Fields page to create them.

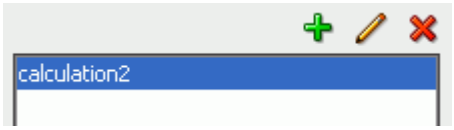
To create calculated fields:

1. Click **Add** to add a calculated field.

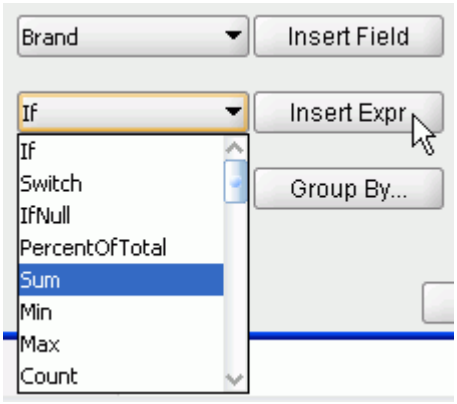
   Click the **Add** icon above and to the right of the box.



   The new default field name appears in the list of calculations. You can rename it later, after entering a valid expression.



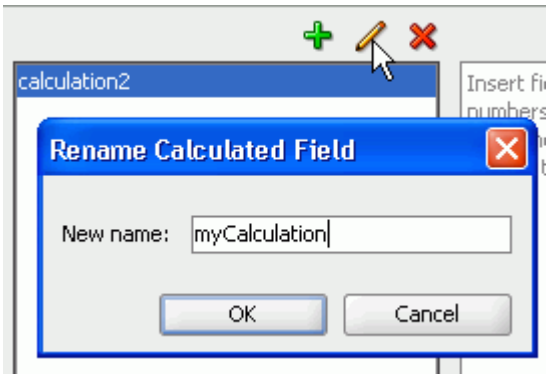
2. To enter an expression, choose an expression from the expressions list, and click **Insert Expr**.



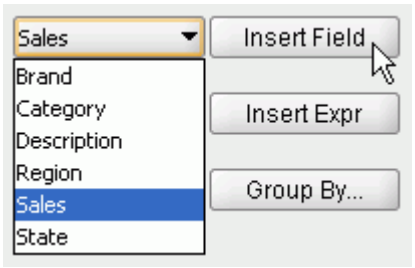
   Complete the expression in the right-hand box, and click **Validate** to check the syntax of your expression.

   There are several preformed expressions available. See *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for examples and more information about each expression.

3. Click **Rename** to change the display name of a calculated field.



4. To use a data object field in a calculation, select the field from the field list, and click **Insert Field**.



51.5.4.1 Creating Groups in Calculated Fields

You can create groups in the calculations page.

To create groups on calculations:

1. Select a calculation in the calculations list.
2. Click **Group By**.

3. Choose a field to group by, and click **OK**.

   You can use the up and down arrows to change the group order.

51.5.5 How to Select, Organize, and Sort Fields

To deselect all of the fields, uncheck the **ALL** checkbox, and select individual fields.

The field at the top of the list appears in the left-most column of the final table in the ADF page. To change the order in which the fields appear, select a field and use the blue arrows to move it up or down the list.

To apply sorting on a field, click the sorting type in the **Sorting** column, and choose a new sorting type from the list.

> **Note:** If you use Active Data, sorting is preserved on Update, Upsert operations, but not on Insert operations.

51.5.6 How to Create Filters

You can apply filters to both Group Query and Flat Query types of Oracle BAM data controls. Add combinations of entries and headers to create complex filter expressions.

51.5.6.1 How to Create Filter Headers

To create a sub-header under an existing header:

1. In the Filters page of the Create BAM Data Control wizard, select a header under which to add the sub-header, and click **Add Header**.

   You can select the main header at the top of the filter expression to create a sub-header under it.

2. To change the operator (default **ALL**), select the header, and click **Edit**. For the following operator options, data is returned when:

   - **ALL**. All of the included entries are true.
   - **NONE**. None of the included entries are true.
   - **AT LEAST ONE**. At least one and maybe more of the included entries are true.
   - **NOT ALL**. Some or none of the included entries are true, but not all of the included entries are true.

3. Select an operator from the **Filters** list, and click **OK**.

51.5.6.2 How to Create Filter Entries

To create a filter entry:

1. In the Filters page of the Create BAM Data Control wizard, select a header under which to add the filter entry.

   For information about creating headers in the filter expression see Section 51.5.6.1, "How to Create Filter Headers."

2. Click **Add Filter Entry**.

   The Add Filter Entry dialog opens.

3. Choose a field from the **Field** list.

4. Choose an expression from the **Comparison** list. Choices include:

   - **is equal to** returns rows containing an exact value match.

     See Section 51.5.6.3, "Entering Comparison Values" for information on configuring comparison values.

   - **is not equal to** returns rows containing all values except specified value.
   - **is less than** returns rows containing values less than specified value.
   - **is less than or equal to** returns rows containing values less than or equal to specified value.
   - **is greater than** returns rows containing values greater than specified value.
   - **is greater than or equal to** returns rows containing values greater than or equal to specified value.
   - **is like** returns rows containing values that match a string pattern. Include an underscore (_) as a wildcard for a single character in a string and a percent symbol (%) as a wildcard for one character or more. Wildcard characters can be combined, for example, %mm _00 would return all columns (35mm 200, 35mm 400, 35mm 800). Do not enter any spaces in the expression since spaces are treated as characters in the data match.
   - **is not like** returns rows containing values that do not match a string pattern.
   - **is null** returns rows containing values where the column is null. If you select this comparison, your filter configuration is complete. Click **OK** to create the filter. For numeric data types, nulls are not returned for filters returning values equal to zero. In other words, zeroes are not treated as null values. A null represents missing data in the field.
   - **is not null** returns rows containing values where the column is not null. If you select this comparison, your filter configuration is complete. Click **OK** to create the filter. For numeric data types, nulls are not returned for filters returning values equal to zero. In other words, zeroes are not treated as null values. A null represents missing data in the field.
   - **is in list** returns rows containing values included in a list. To build a list, click **Edit**. Type a value in the field and click **Add** to add it to the list. Add as many values as needed. Click **Browse** to choose values currently present in the Data Object. Click **Remove** to remove a value. Click **OK** to close the dialog.
   - **is not in list** returns rows containing values not included in the list. To build a list, click **Edit**. Type a value in the field and click **Add** to add it to the list. Add as many values as needed. Click **Browse** to choose values currently present in

the Data Object. Click **Remove** to remove a value. Click **OK** to close the dialog.

- **is within a time interval** returns rows containing values that occur within the specified time interval. Configure the time interval using the provided lists. Select a **Type**, enter a multiplier in the field and select a **Unit**.

  When filtering on a datetime or timestamp field, you can enable **Active Now** to keep the displayed time interval current as time passes. Configure the **Active Now Interval** to specify how often to refresh the display. See Section 51.5.6.4, "Using Active Now" for more information.

- **is within the current time period** returns rows containing values that occur within the current specified time unit. Select a **Unit** from the list.

  When filtering on a datetime or timestamp field, you can enable **Active Now** to keep the displayed time period current as time passes. See Section 51.5.6.4, "Using Active Now" for more information.

- **is within a time period** returns rows containing values that occur within the specified time period. Configure the time period using the provided lists. Enter a value in the **Offset** field, select a **Unit**, and select a **Type**.

  When filtering on a datetime or timestamp field, you can enable **Active Now** to keep the displayed time period current as time passes. See Section 51.5.6.4, "Using Active Now" for more information.

5. Click **OK** to add the entry to the filter expression.

51.5.6.3 Entering Comparison Values

For most **Comparison** values you must choose **Value**, **Field**, or **Calculation** from the **Value** list.

Only the following comparisons do not require a comparison value:

- is null
- is not null
- is in list
- is not in list
- is within a time in interval
- is within the current time period
- is within a time period

51.5.6.3.1 Comparison With a Value If you select **Value**, do one of the following:

- Click **Browse** to see a list of values present in the data object. Select a value from the list. Up to 50 values display in the list. The field can be left blank to create a filter on a blank string.

> **Note:** If there are more than 50 values in the field, not all of the values are shown in the **Browse** list. Your Oracle Business Activity Monitoring administrator can configure the number of rows to display in the list. See the *Oracle Business Activity Monitoring Installation Guide* for more information.

- Manually enter a value in the field.

51.5.6.3.2 Comparison With a Calculation If you select **Calculation**, enter an expression in the field to compare with the first field.

For example, if you create a list view using the sample Call Center data object and create a filter with the following attributes:

- **Field.** Total
- **Comparison**. is equal to
- **Value**. Calculation
- **Calculation field**. Quantity*2

This filter yields only those rows where the value in the Total column is equal to twice the value in the Quantity column.

51.5.6.3.3 Comparison With a Field If you select **Field**, select a field from the last list to compare with the field selected in the **Field** list.

51.5.6.3.4 Comparison with a Parameter If you select **Parameter**, select a parameter from the list at the right. Creating a filter using a parameter allows the user to change the filter values at runtime.

The list contains the parameters you created in the Parameters step of the Create Oracle BAM Data Control wizard. For more information about creating parameters see Section 51.5.2, "How to Create Parameters."

51.5.6.4 Using Active Now

The Active Now feature in data filtering enables you to display in your views a segment of the data that is always within a defined time window. As time passes, the view is updated with the data within the defined time interval in the filter. Older data is removed from the view and newer data is added as time passes.

Active Now is available when you choose one of the following comparison expressions:

- **is within a time interval**
- **is within the current time period**
- **is within a time period**

Active Now behaves differently depending on which comparison expression you choose.

When you choose **is within a time interval**, you can control how often the data is refreshed using the **Active Now Interval** setting.

For example, if you create a filter using **is within a time interval**, **previous** type, **1**, **Hours** unit, and **Active Now**, set the **Active Now Interval** to 60 seconds, and the current time is 3:25 p.m., data from 2:25 p.m. - 3:25 p.m. is displayed in the view. When the current time changes to 3:26 p.m., data from 2:26 p.m. - 3:26 p.m. is displayed in the view. Every 60 seconds the oldest minute of data is removed from the view and the newest minute is added.

When you choose **is within the current time period** or **is within a time period**, the data is refreshed when the time period changes.

For example, when you create a filter using **is within the current time period**, the **Hours** unit, and **Active Now**, and the current time is 3:25 p.m., only data from 3:00

p.m. - 3:59 p.m. is displayed in the view until the current time is 4:00 p.m. At 4:00 p.m. all the data from 3:00 p.m. - 3:59 p.m. is removed from the view, and data that accumulates during the 4:00 p.m. - 4:59 p.m. time interval is displayed in the view.

51.5.7 How to Select and Organize Groups

To specify a group:

1. In the Groups page of the Create BAM Data Control wizard, select one or more fields in the **Group Fields** list.

   If you created a Field parameter, it appears in the list. See Section 51.5.2, "How to Create Parameters" for more information about creating field parameters.

   To group by numeric fields, first select **Show Numeric Fields** at the bottom of the list.

2. To change the display order in which the groups are presented in a graph, select a sorting option from the **Sorting** list for any selected field.

3. If a datetime field is selected in the **Fields** list, several options are enabled for configuring Time Groups on the right side of the wizard page.

   See Section 51.5.7.1, "How to Configure Time Groups and Time Series" for more information.

51.5.7.1 How to Configure Time Groups and Time Series

You can create a chart where the grouping (x axis) is based on a datetime field.

To configure time groups:

1. In the Groups page of the Create BAM Data Control wizard, select a single field of type datetime in the **Group Fields** list.

   This action enables the Time Groups options on the right side of the wizard page.

2. Select **Continuous Time Series** to display empty groups for time intervals where no data is available.

   There may be time gaps where the data object did not have entries. The Continuous Time Series feature adds groups to the result whose values are zero, so that when the results are shown on the graph, the x axis represents a smooth time series.

   Continuous Time Series is valid only if you have chosen a single datetime field to group by. Continuous Time Series is not supported if any additional group fields are selected.

3. Select either **Use Time Series** or **Use Time Groups**.

   - **Use Time Series** displays the data from the first datetime data point available in the data object to the last in the configured time interval.
   - **Use Time Groups** displays data grouped into a set number of time intervals. For example, if you select Month from the time unit list, all data from January from all years where data is available are grouped in one data point on the chart.

4. Select a time unit from the list.

   If you selected **Use Time Groups**, the groups are described in the following list.

   – **Year** displays groups for all of the years where data is available.

- **Quarter** displays four groups representing the quarters of a year (January-March, April-June, July-September, and October-December).
- **Month** displays twelve groups representing the months of the year.
- **Week** displays 52 groups representing the weeks in a year.
- **Day of Year** displays groups representing the 365 possible days in a year.
- **Day of Month** displays 31 groups representing the possible days of a month.
- **Day of Week** displays seven groups representing the days of the week.
- **Hour** displays 24 groups representing the hours of a day.
- **Minute** displays 60 groups representing the minutes in an hour.
- **Second** displays 60 groups representing the seconds in a minute.

5. Enter a quantity of the time unit to group by. For example, entering a **2** next to the Month time unit displays the groups in two month increments (January and February are grouped as one data point on the chart).
6. Click **Next** or **Finish**.

51.5.8 How to Create Aggregates

To specify an aggregate on a field:

1. In the Aggregates page of the Create BAM Data Control wizard, select a field in the **Fields** list.

   The valid **Summary Functions** for the data type of that field are enabled.

2. Select one or more valid **Summary Functions**.

   The expressions appear in the Summary Values list.

51.5.9 How to Modify the Query

To edit the Oracle BAM data control query, right click the data control node, and select **Edit Definition**. The Edit BAM Data Control wizard opens and you can jump to any page to edit that part of the query.

51.6 Using Oracle BAM Data Controls in ADF Pages

Oracle BAM data controls can be used in all ADF Faces components. Only a subset of ADF Faces components are ADS (active data service) capable. Refer to *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* for information about ADF Faces components that support ADS.

Oracle BAM data control instances use the resources of the Oracle BAM Server instance they are connected to. Those resources are released when the data control is released. In order to release those resources in a timely fashion it is required that you use Oracle BAM data controls within bounded ADF task flows with Data Control Scope set to `isolated`. It is recommended that you set the session time out in `web.xml` to a reasonable value so that resources are released in a timely way. Refer to *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* for information about the general life cycle of data controls.

Note: Oracle BAM data control instance sharing is not supported. When two or more ADF Faces components must display the same data, and are bound to the same Oracle BAM data control definition, make sure to wrap each ADF Faces component in a bounded ADF task flow, and set the Data Control Scope to `isolated`. See *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* for more information.

When using an Oracle BAM data control with `setParamters` inside the task flow, you must pass the parameter to the task flow and ensure that the method gets called before the query call.

51.6.1 How to Use an Oracle BAM Data Control in a JSF Page

To use an Oracle BAM data control in a JSF page:

1. Set the default browser:
   a. In the JDeveloper Tools menu, select **Preferences**.
   b. In the Preferences dialog, select **Web Browser and Proxy**.
   c. Choose a default browser by entering the path to the browser's executable in the **Browser Command Line** field, enter any applicable proxy information, and click **OK**.
2. Create a bounded ADF Task Flow:
   - Right click project, select **New**, select **JSF** under the Web Tier category, and select **ADF Task Flow** in Items list.
   - Make sure the **Create as Bounded Task Flow** option is checked.
   - Drag and drop **View** from Components in the Component Palette to the ADF Task Flow editor.
   - Drag and drop **data-control-scope** under Source Elements in the Component Palette to the ADF Task Flow editor.
   - Change the **Data Control Scope** to `isolated` in the Property Inspector.
3. Create a JSF Page Fragment by double-clicking in the View you previously dropped in the ADF Task Flow editor.
4. Drag and drop an accessor node from the Data Controls panel to the JSF page editor.
5. Select a data visualization component.

   A subset of ADF components support active data. See the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* for more information about binding data controls with data visualization components.
6. Save all, and in the Oracle JDeveloper toolbar, click **Run Project**.

51.7 Deploying Applications With Oracle BAM Data Controls

At runtime, Oracle BAM data control must use the Oracle BAM connection to connect to Oracle BAM Server.

Deployment to the Integrated WebLogic Server is automatic; however, deployment to a standalone Oracle WebLogic Server requires some extra steps. See Section 51.7.1,

"How to Deploy to Oracle WebLogic Server in Development Mode," and Section 51.7.2, "How to Deploy to a Production Mode Oracle WebLogic Server," for more information

See *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* for more information about deploying Fusion Web applications.

51.7.1 How to Deploy to Oracle WebLogic Server in Development Mode

Before deployment to a development-mode Oracle WebLogic Server, verify that the Java system property `jps.app.credential.overwrite.allowed` to `true` during Oracle WebLogic Server startup.

Add the following to the `JAVA_PROPERTIES` entry in the *`ORACLE_HOME`*`/user_projects/domains/`*`domain`*`/bin/setDomainEnv.sh` file:

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

Post-deployment configuration is not required.

51.7.2 How to Deploy to a Production Mode Oracle WebLogic Server

Before and after deploying an ADF application with Oracle BAM data controls to a production-mode Oracle WebLogic Server you must do the following steps:

1. The application must be deployed using MDS (Metadata Data Services). To enable MDS:
   a. Create and/or register your MDS Repository.
   b. Edit the `adf-config.xml` file to add the following block:

```
<adf-mds-config xmlns="http://xmlns.oracle.com/adf/mds/config">
  <mds-config xmlns="http://xmlns.oracle.com/mds/config"
version="11.1.1.000">
    <persistence-config>
      <metadata-store-usages>
        <metadata-store-usage default-cust-store="true"
deploy-target="true" id="myRepos">
        </metadata-store-usage>
      </metadata-store-usages>
    </persistence-config>
  </mds-config>
</adf-mds-config>
```

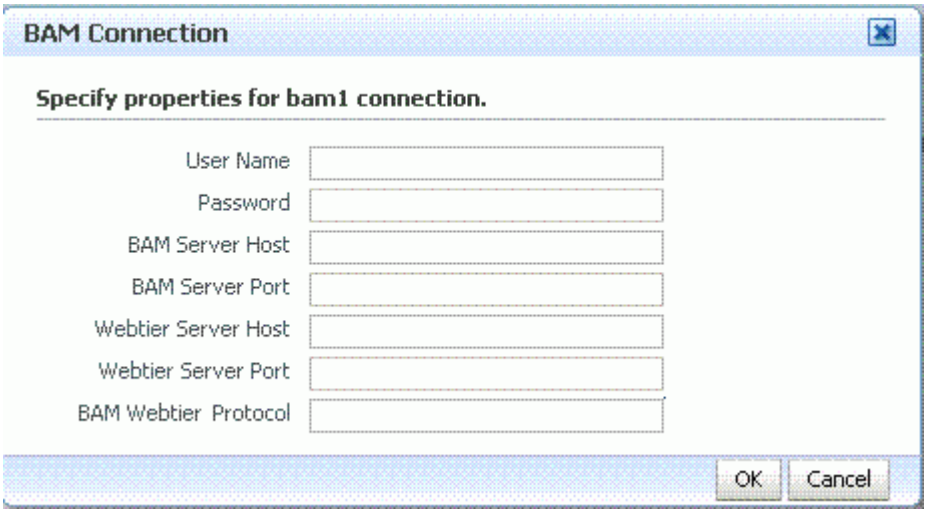
   c. Deploy the application to Oracle WebLogic Server after choosing the appropriate repository during deployment from the MDS Repository dialog that opens.
2. After deployment, the Oracle BAM connection must be re-created in Oracle Enterprise Manager Fusion Middleware Control.

   Go to the ADF Connections Configuration page, and create a **BAM** connection.
   a. Open Oracle Enterprise Manager Fusion Middleware Control (`http://`*`host`*`:`*`port_number`*`/em`).
   b. In the left pane select **Deployments**, then select your application.
   c. In the right pane, select **Application Deployment** and select **ADF -> Configure ADF Connections** in the menu.
   d. Select **BAM** in the **Connection Type** list.

e. Enter the **Connection Type** to be the same as the one defined in Oracle JDeveloper.

f. Select **Create Connection** to add a new row under **BAM Connections**.

g. Select the new connection and click the **Edit** icon, specify the appropriate values for all connection parameters in the dialog, and click **OK**.



The Oracle BAM Web Tier is the location where report server is running. The valid values for **BAM Webtier Protocol** are `http` and `https`.

You must enter the same **Connection Name** as the Oracle BAM connection that was configured for design time (see Section 50.4, "Creating a Design Time Connection to an Oracle BAM Server").

h. Click **Apply** and re-run the page.

52

Defining and Managing Oracle BAM Data Objects

This chapter contains the information needed to create and manage data objects, including assigning permissions, managing folders, creating security filters, and adding dimensions and hierarchies.

This chapter includes the following sections:

52.1 Introduction to Oracle BAM Data Objects

Data objects are tables that store raw data in the database. Each data object has a specific layout which can be a combination of data fields, lookup fields, and calculated fields.

The data objects are used to create reports in Oracle BAM Active Studio, active data visualization components in ADF applications, among other uses. For more information about how data objects are used see "Creating and Managing Reports" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* and Chapter 51, "Using Oracle BAM Data Control."

The data objects you define are based on the types of data available from Enterprise Message Sources (EMS) that you can define in Oracle BAM Architect. You must define columns in the data object. The data object contains no data when you create it. You must load or stream data into data objects using the technologies discussed in the following topics:

- Chapter 50, "Integrating Oracle BAM with SOA Composite Applications"

- Chapter 53, "Creating Oracle BAM Enterprise Message Sources"
- Chapter 54, "Using Oracle Data Integrator With Oracle BAM"
- Chapter 55, "Creating External Data Sources"
- Chapter 56, "Using Oracle BAM Web Services"

Data objects can also be accessed and updated by Oracle BAM alerts. See Chapter 57, "Creating Oracle BAM Alerts" for more information.

WARNING: Do not read or manipulate data directly in the database. All access to data must be done using Oracle BAM Architect or the Oracle BAM Active Data Cache API.

52.2 Defining Data Objects

Data objects are defined using Oracle BAM Architect. See the following topics for more information:

- Section 52.2.1, "How to Define a Data Object"
- Section 52.2.2, "How to Add Columns to a Data Object"
- Section 52.2.3, "How to Add Lookup Columns to a Data Object"
- Section 52.2.4, "How to Add Calculated Columns to a Data Object"
- Section 52.2.5, "How to Add Time Stamp Columns to a Data Object"
- Section 52.2.6, "What You May Need to Know About System Data Objects"
- Section 52.2.7, "What You May Need to Know About Oracle Data Integrator Data Objects"

52.2.1 How to Define a Data Object

To define a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Click **Create Data Object**.
3. Enter a name for the data object.

   Caution: A single or double quotation mark in an Oracle BAM object name, such as a data object, report, or enterprise message source name, causes a runtime error.

   Do not include single or double quotation marks in an Oracle BAM object name.

4. Enter the path to the location in the folder tree in which to store the data object. Click Browse to use the Select a Folder dialog.
5. Optionally, enter a description of the data object.
6. If this data object is loaded from an External Data Source (EDS) select the **External Data Source** checkbox and configure the following:

- Select an **External Data Source** from the list. EDS definitions are configured on the External Data Sources screen. See Chapter 55, "Creating External Data Sources" for more information.
- Select the **External Table Name**.

Note: Only the tables that belong to the user are shown when a data object is created on an EDS.

Creating a data object with multiple time stamp fields on an EDS is not supported.

---

7. Add columns to the data object using the **Add a field** or **Add one or more lookup fields** options.

   See Section 52.2.2, "How to Add Columns to a Data Object" and Section 52.2.3, "How to Add Lookup Columns to a Data Object" for more information.

8. Click **Create Data Object** when you are finished adding columns or lookup columns.

52.2.2 How to Add Columns to a Data Object

To add columns to a data object:

1. In a data object you are creating or editing, click **Add a field**.
2. Specify the column name, data type, maximum size (scale for decimal columns), whether it is nullable, whether it is public, and tip text.

   If you are adding a column in a data object based on an External Data Source you must also supply the **External field name**.

   The data types include:

   - **String.** Text columns containing a sequence of characters.

     A string with a max size greater than 0 and less than or equal to 2000 becomes an Oracle database data type VARCHAR field. If the max size is less than zero or greater than 2000 the string field is stored as a CLOB. To get a CLOB field, just define a string field with a max size greater than 2000.

   - **Integer.** Numeric columns containing whole numbers from -2,147,483,648 to 2,147,483,648.
   - **Float.** Double-precision floating point numbers.

     The Oracle BAM Float type does not map to the Oracle database Float type. Oracle BAM Float truncates numeric data that has very high precision. If you do not want to see loss of precision use the Oracle BAM Decimal type (`NUMBER` in Oracle database) with the scale you want.

   - **Decimal.** Numbers including decimal points with scale number defined. The number is stored as a string which uses a character for each digit in the value.

     The Oracle BAM Decimal data type is stored as a `NUMBER (38, X)` in the Oracle database. The first argument, `38`, is the precision, and this is hard-coded. The second argument, `X`, is the scale, and you can adjust this value. The scale value cannot be greater than 38.

   - **Boolean.** Boolean columns with true or false values.

- **Auto-incrementing integer.** Automatically incremented integer column.
- **DateTime.** Dates and times combined as a real number.
- **Timestamp.** Date time stamp generated to milliseconds. A data object can contain only one time stamp field. See Section 52.2.5, "How to Add Time Stamp Columns to a Data Object" for more information.

  A DateTime field is stored as an Oracle database data type DATE. A Timestamp field is stored as an Oracle database data type TIMESTAMP(6). Depending on how the Timestamp field is populated, Oracle BAM may fill in the time stamp value for you. For instance, in Oracle BAM Architect you cannot specify the value for Timestamp when adding a row, but if the value for Timestamp is specified in an ICommand import file, the specified value is added as the value of Timestamp instead of the current time.
- **Calculated.** Calculated columns are generated by an expression and saved as another data type. See Section 52.2.4, "How to Add Calculated Columns to a Data Object" for more information.

Keep adding columns using **Add a field** and **Add one or more lookup fields** until all the required columns are listed. Click **Remove** to remove a column in the data object.

3. Click **Save changes**.

52.2.3 How to Add Lookup Columns to a Data Object

You can add lookup columns to a data object. This performs lookups on key columns in a specified data object to return columns to the current data object. You can match multiple columns and return multiple lookup columns.

To add a lookup column to a data object:

1. In a data object you are creating or editing, click **Add one or more lookup fields**.

   The Define Lookup Field dialog opens.
2. Select the data object to use for the lookup.
3. Select the lookup columns from the data object. You can select one or more columns by holding down the Shift or Control key when selecting. Selecting multiple columns creates multiple lookup columns in the data object. These are the columns you want to return.
4. Select the column to match from the lookup data object.
5. Select the column to match from the current data object. You must have previously created other columns in this data object so that you have a column to select.
6. Click **Add**.

   The matched column names are displayed in the list. You can click **Remove** to remove any matched pairs you create.
7. You can repeat steps 4 through 6 to create multiple matched columns. This is also known as a composite key.
8. Click **OK** to save your changes and close the dialog.

   The new lookup columns are added to the data object. Click **Modify Lookup Field** in **Layout > Edit Layout page** to make changes to a lookup column. Multiple selection of return columns is possible when defining a new lookup but not when modifying an existing one.

You can click **Remove** to remove any lookups you create.

Note: Oracle Business Activity Monitoring supports two types of schema models: unrelated tables or star schemas. Any other kind of schema that does not conform to these models may result in performance issues or deadlocks. Snowflake dimensions (daisy-chained lookups) are not supported.

Supported:

```
Table 1 (with no lookups to any other tables)
Table 1 > Lookup > Table 2
```

Not supported:

```
Table 1 > Lookup > Table 2 > Lookup > Table 3
```

52.2.4 How to Add Calculated Columns to a Data Object

When creating calculated columns in a data object you can use operators and expression functions, combined with column names, to produce a new column.

Table 52–1 Describes the operators you can use to build calculated columns.

The *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* provides the syntax and examples for expressions you can use in a calculated column.

Numbers of type Decimal require a "D" character suffix when used in a calculated column (field). See *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for more information.

Table 52–1 Operators Used in Calculated Columns

Operator	Function
+ (plus sign)	Add
- (minus sign)	Subtract
* (asterisk)	Multiply
/ (slash)	Divide
% (percent sign)	Modulus
() (parentheses)	Parentheses determine the order of operations
&& (double ampersand)	Logical AND
!= (exclamation point and equal sign)	Logical NOT
|| (double pipe)	Logical OR For example `if ((CallbackClientTime == NULL) || (ReceiveInputTime == NULL)) then (-1) else (CallbackClientTime-ReceiveInputTime)`
== (double equal sign)	Equality
= (equal sign)	Assignment

Column names containing any special characters, such as the operators listed in Table 52–1 double quotation marks, or spaces, must be surrounded with curly braces

{}. If column names contain only numbers, letters and underscores and begin with a letter or underscore they do not need curly braces. For example, if the column name is **Sales+Costs**, the correct way to enter this in a calculation is `{Sales+Costs}`.

Double quotation marks must be escaped with another set of double quotation marks if used inside double quotation marks. For example, `Length("""Hello World, "" I said").`

WARNING: If you enter a calculated column with incorrect syntax in a data object, you could lose the data object definition.

52.2.5 How to Add Time Stamp Columns to a Data Object

You can create a date time stamp column generated to milliseconds by selecting the **Timestamp** data type. This column in the data object must be empty when the data object is populated by the Oracle BAM ADC so that the time stamp data can be created.

52.2.6 What You May Need to Know About System Data Objects

The System data objects folder contains data objects used to run Oracle Business Activity Monitoring. You should not make any changes to these data objects, except for the following:

- **Custom Parameters** lets you define global parameters for Action Buttons.
- **Action Form Templates** lets you define HTML forms for Action Form views.
- **Chart Themes** lets you add or change color themes for view formatting.
- **Matrix Themes** lets you add or change color themes for the Matrix view.
- **Util Templates** lets you define templates that are used by Action Form views to transform content.

For more information about matrix and color themes, Action Buttons, and Action Forms see *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring*.

52.2.7 What You May Need to Know About Oracle Data Integrator Data Objects

If you install the integration files for Oracle BAM and Oracle Data Integrator, three data objects are created in Oracle BAM Architect: Context, Scenarios and Variables in the `/System/ODI/` folder. These data objects should not be deleted from Oracle BAM Architect, and their configuration should not be changed.

52.3 Creating Permissions on Data Objects

You can add permissions for users and groups on data objects. When users have at least a read permissions on a data object they can choose the data object when creating reports.

52.3.1 How to Create Permissions on a Data Object

To add permissions on a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object.

The general information for the data object is displayed in the right frame.

3. Click **Permissions**.
4. Click **Edit Permissions**.

   Alternatively you can copy permissions from another data object. See Section 52.3.3, "How to Copy Permissions from Other Data Objects" for more information.
5. Click the **Restrict access to Data Object to certain users or groups** checkbox.

   The list of users and groups and permissions is displayed.
6. You can choose to display the following by choosing an option:
   - Show all users and groups
   - Show only users and groups with permissions
   - Show users only
   - Show groups only
7. You can set permissions for the entire list by clicking the buttons at the top of the list.

   The permissions are Read, Update, and Delete. You can set permissions for individual users or groups in the list by clicking the checkbox in the permission column that is next to the user or group name.

   Note: Delete and Update permissions are not effective unless a user is also granted the Read permission.

   Members of the Administrator role have all permissions to all data objects, and their permissions cannot be edited.
8. After indicating the permissions with selected checkboxes, click **Save changes**.

   A message is displayed to confirm that your changes are saved.
9. Click **Continue** to display the actions for the data object.

52.3.2 How to Add a Group of Users

Users assigned to the Administrator role have access to all data objects. The Administrator role overrides the data object permissions.

To add a group to the list:

1. Click **Add a group to the list**.
2. Type the Windows group name in the field. The group must previously exist as a domain group.
3. Click **OK**.

   The group is added to the list.

52.3.3 How to Copy Permissions from Other Data Objects

You can copy the permissions from another data object and then make additional changes to the permissions before saving.

In Oracle BAM Architect for a data object, click Permissions and then click Copy from. Select the data object that contains the permissions to copy and click OK. You can edit the copied permissions and click **Save changes**.

To copy permissions from another data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Click the data object to add a security filter to.

   The general information for the data object is displayed in the right frame.
3. Click **Permissions**.
4. Click **Copy from**.

   The Choose Data Object dialog opens.
5. Select the data object that contains the permissions to copy and click **OK**.
6. If the data object previously had no permissions assigned, select the **Restrict access to Data Object** checkbox.
7. You can edit the copied permissions or add a group to the list.
8. Click **Save changes**.

52.4 Viewing Existing Data Objects

This section describes how to view information about data objects.

52.4.1 How to View Data Object General Information

The general information of a data object displays the owner, when it was created, when it was last modified, and the row count.

To view the general information of a data object:

- Click the data object in the list.

  If you are currently viewing the layout or contents of a data object, click *General*.

  The general information is displayed in the right frame. It contains the following information:

  - **Created**. Date and time the data object was created.
  - **Last modified**. Date and time the data object was last modified.
  - **Row count**. Number of rows of data in the data object.
  - **Location**. Location of the data object.
  - **Type**. Type of the data object.
  - **Data Object ID**. The ID used to identify the data object. This is based on the name although the ID is used throughout the system so that you can edit the name without affecting any dependencies.

Note: If the row count is over 500,000 rows, an approximate row count is displayed in the General information for increased performance purposes. The approximate row count is accurate within 5-10% of the actual count. If you want to view an exact row count instead of the approximation, click Show exact count. The exact count is displayed. This could take a few minutes if the data object has millions of rows.

52.4.2 How to View Data Object Layouts

The layout describes the columns in a data object. The columns are described by name, column ID, data type, maximum length allowed, scale, nullable, public, calculated, text tip, and lookup.

To view the layout of a data object:

1. Select the data object.
2. The general information is displayed in the right frame.
3. Click **Layout**.

   The layout information is displayed in the right frame. It contains the following information:

   - **Field name**. Name of the column.
   - **Field ID**. Generated by the system.
   - **External name**. External column name from the External Data Source (only appears in data objects based on External Data Sources).
   - **Field type**. Data type of the column.
   - **Max length**. Maximum number of characters allowed in column value.
   - **Scale**. Number of digits on the right side of the decimal point.
   - **Nullable**. Whether the data type can contain null values.
   - **Public**. This setting determines if the column is available in Oracle BAM Active Studio to use in a report. If the box is unchecked, the column does not appear in Oracle BAM Active Studio. This is useful for including columns for calculations in data objects that should not appear in reports.
   - **Lookup**. Displays specifics of a lookup column.
   - **Calculated**. Displays the expression of a calculated column.
   - **Tip Text**. Helpful information about the column.

52.4.3 How to View Data Object Contents

You can view the rows of data stored in a data object by viewing the data object contents. You can also edit the contents of the data object.

To view the contents of a data object:

1. Select the data object.

   The general information is displayed in the right frame.

2. Click **Contents**.

The first 100 rows of the data object display in the right frame.

(To change this default, update the `Architect_Content_PageSize` property. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for information.)

Oracle BAM Architect displays the total number of rows in the data object and the number of rows that are available for viewing. For better server performance, the number of rows shown in Oracle BAM Architect is limited by configuration properties.

When internal data objects are displayed in **No row number** mode (default), you can view all of the records in the data object using the navigation tools.

When internal data objects are displayed in **Show row numbers** mode, you can view a limited number of records. This number is 64000 by default, and can be changed by modifying the `ADCMaxViewsetRowCount` property in `BAMServerConfig.xml`.

When external data objects are displayed in either mode, you can view a limited number of records. This number is 64000 by default, and can be changed by modifying the `Import_Maxsize` property in `BAMServerConfig.xml`.

3. Click **Next**, **Previous**, **First**, and **Last** to go to other sets of rows.

   Rows are listed with a Row ID column. Displaying only Row ID provides faster paging for large data objects. Row IDs are assigned one time in each row and maintain a continuous row count when you clear and reload a data object.

   You can click **Show row numbers** to display an additional column containing a current row count starting at 1. Click **No row numbers** to hide the row count column again.

4. Click **Refresh** to get the latest available contents.

52.5 Using Data Object Folders

You can organize data objects by creating folders and subfolders for them. When you create a folder for data objects, you can assign permissions by associating users and actions with the folder.

52.5.1 How to Create Folders

You can create new folders for organizing data objects. Then you can move or create data objects into separate folders for different purposes or users. After creating folders, you can set folder permissions to limit which users can view the data objects it contains.

To create a new folder:

1. Select **Data Objects** from the Oracle BAM Architect function list.

   The current data object folders display in a tree hierarchy.

2. Click **Create subfolder**.

   A field for naming the new folder is displayed.

3. Enter a name for the folder and click **Create folder**.

   The folder is created as a subfolder under the Data Objects folder and a message is displayed confirming that the new folder was created.

4. Click **Continue** to view the folder.

52.5.2 How to Open Folders

To open a folder:

1. Expand the tree of folders by clicking the + (plus sign) next to the Data Objects folder.

   The System subfolders contain data objects for running Oracle Business Activity Monitoring. For more information about these data objects see Section 52.2.6, "What You May Need to Know About System Data Objects."

2. Click the link next to a folder to open it.

   The folder is opened, and the data objects in the folder are shown in the list underneath the folder tree. The general properties for the folder display in the right frame and the following links apply to the current folder:

   View. Displays the general properties of this folder such as name, date created, date last modified, user who last modified it. View is selected when you first click a folder.

   Create subfolder. Creates another folder within the selected folder.

   Delete. Removes the selected folder and all the data objects it contains.

   Rename. Changes the folder name.

   Move. Moves this folder to a new location, for example, as a subfolder under another folder.

   Permissions. Sets permissions on this folder.

   Create Data Object. Creates a data object in this folder.

52.5.3 How to Set Folder Permissions

When you create a folder, you can set permissions on it so that other users can access the data objects contained in the folder.

To set permissions on a folder:

1. In the Data Objects folder, select the folder to change permissions on.
2. Click **Permissions**.
3. Click **Edit permissions**.
4. Select the **Restrict access to Data Object to certain users or groups** checkbox.

   The list of users and groups and permissions is displayed.

5. You can choose to display the following by selecting an option:
   - Show all users and groups
   - Show only users and groups with permissions
   - Show users only
   - Show groups only
6. You can set permissions for the entire list by clicking the column headers at the top of the list.

The permissions are Read, Update, and Delete. You can set permissions for individual users or groups in the list by selecting the checkbox in the permission column that is next to the user or group name.

Note: Delete and Update permissions are not effective unless a user is also granted the Read permission.

7. After indicating the permissions with selected checkboxes, click **Save changes**.

   A message is displayed to confirm that your changes are saved.

8. Click **Continue** to display the actions for the data object.

To add a group to the list:

1. Click the **Add a group to the list** link.
2. Type the Windows group name in the field. The group must previously exist as a domain group.
3. Click **OK**.

   The group is added to the list.

52.5.4 How to Move Folders

To move a folder:

1. Select the folder to move.
2. Click **Move**.
3. Click **Browse** to select the new location for the folder.
4. Click **OK** to close the dialog.
5. Click **Move folder**.

   The folder is moved.

6. Click **Continue**.

52.5.5 How to Rename Folders

To rename a folder:

1. Select the folder to rename.
2. Click **Rename**.
3. Enter a new name and click **Rename folder**.

   The folder is renamed. You must assign unique folder names within a containing folder.

4. Click **Continue**.

52.5.6 How to Delete Folders

When you delete a folder, you also delete all of the data objects in the folder.

To delete a folder:

1. Select the folder to delete.
2. Click **Delete**.

   A message is displayed to confirm deletion of the folder and all of its contents.
3. Click **OK**.

   The folder is deleted.
4. Click **Continue**.

52.6 Creating Security Filters

You can add security filters to data objects so that only specific users can view specific rows in the data object. This can be useful when working with data objects that contain sensitive or confidential information that is not intended for all report designers or report viewers.

Security filters perform a lookup using another data object, referred to as a security data object, containing user names or group names. Before you can add a security filter, you must create a security data object containing the user names or group names and the value in the column to allow for each user name or each group name. Security data objects cannot contain null values.

If the user has a view open, and you change that user's security filter, it does not effect the currently open view. If the user reopens that view, it has the new security filter settings applied. Security filter settings are used to construct the query behind the view at view construction time, so changes to a security filter are not seen by previously created views.

52.6.1 How to Create a Security Filter

To add a security filter to a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to add a security filter to.

   The general information for the data object is displayed in the right frame.
3. Select **Security Filters**.

   If the data object includes security filters, the filter name is displayed and you can expand and view the information.
4. Click **Add filter**.

   The fields for defining the security filter display.
5. Enter the following information:

   Name of this Security Filter. Type a name for this filter.

   Security Data Object. Select the name of the security data object containing the mapped columns.

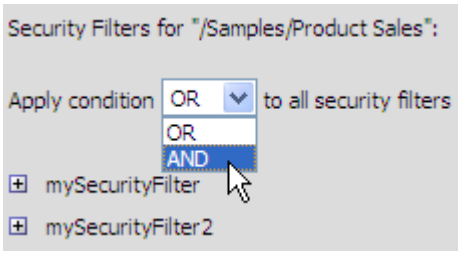
   Type of identification. Select either By user or By group from the dropdown list. The security data object must include either domain or local users or groups mapped to values in the identification column.

   Identification column in Security Data Object. Select the name of the column for containing user names or group names.

Match column in Security Data Object. Select the column to match in the security data object.

Match column in this Data Object. Select the name of the column to match in this data object.

6. Click **Add**.
7. Select the **OR** or **AND** condition when multiple security filters are created on a data object.



By default the security filters are applied with an OR condition, meaning that if there is a match in one security data object, then the user or group identified can access the data object. The AND condition requires that the user or group be identified in all of the security data objects to access the data object protected by the filters.

Note: If there are more than two security filters, you cannot use both AND and OR. You must either use AND or OR for all of the filters.

Security Filters: An Example

For example, to add a security filter to the following data object, you need a security data object containing Region information to perform the security lookup.

Sample data object:

User	Region	Sales
John Smith	1	$55,000
Bob Wright	1	$43,000
Betty Reid	2	$38,000

Security data object:

Login	Region ID
DomainName\jsmith	1
DomainName\jsmith	2
DomainName\bwright	1
DomainName\breid	2

When the **bwright** account views a report that accesses the data object with a security filter applied based on Region ID and Region, it is only able to access information for jsmith and bwright. It is not able to view the breid information because it is not able to

view data for the same region. However, the jsmith account is set up to view data in both region 1 and 2.

52.6.2 How to Copy Security Filters from Other Data Objects

You can copy security filters from another data object and apply them to the data object you are editing.

To copy security filters from another data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to add a security filter to.

   The general information for the data object is displayed in the right frame.
3. Select **Security Filters**.

   If the data object includes security filters, the filter name is displayed and you can expand and view the information.
4. Click **Copy from**.

   The Choose Data Object dialog opens.
5. Select the data object that contains the security filters to copy and click **OK**.
6. You can make changes to the security filters by viewing the filter details and clicking **Edit**.
7. Click **Save**.

52.7 Creating Dimensions

In Oracle BAM Architect, you can add dimensions to data objects to define drill paths for charts in Oracle BAM Active Studio. Dimensions contain columns in a hierarchy. When a hierarchy is selected in chart, the end user can drill down and up the hierarchy of information. When a user drills down in a chart, they can view data at more and more detailed levels.

Hierarchies are an attribute of a dimension in a data object. Multiple dimensions can be created in each data object. Each column in a data object can belong to one dimension only. You can create and edit multiple, independent hierarchies.

To use hierarchies as drill paths in charts, the report designer must select the hierarchy to use as the drill path. To create a dimension, you must select multiple columns to save as a dimension. Then you organize the columns into a hierarchy.

The following is a sample dimension and hierarchy:

Dimension	Hierarchy
Sales	Category
	Brand
	Description

52.7.1 How to Create a Dimension

To add a dimension and hierarchy:

1. Select **Data Objects** from the Oracle BAM Architect function list.

2. Select the data object to add a dimension to.

   The general information for the data object is displayed in the right frame.

3. Select **Dimensions**.

4. Click **Add a new dimension**.

5. Enter a dimension name.

6. Enter a description for the dimension. A description is required for drilling configuration.

7. Select the column names to include in the dimension. An example is Sales, Category, Brand, and Description.

   The column names are moved from the Data Objects Fields list to the Dimension Fields list to show that they are selected.

8. Click **Save**.

9. Click **Continue**.

   The new dimension is listed. You must still define a hierarchy for the columns.

10. Click **Add new hierarchy**.

11. Enter a hierarchy name.

12. Enter a description for the hierarchy.

13. Select the column names to define as attributes for the dimension. An example is Sales remains in the Dimension Field list, and you click Category, Brand, and Description to arrange them in a general to more specific order. The order that you click the columns is the order that they are listed in the Hierarchy Field list. Arrange the more general grouping column at the top of the Hierarchy Fields list and the most granular column at the bottom of the Hierarchy Fields list.

14. Click **Save**.

15. Click **Continue**.

    The new hierarchy is listed. You can edit or remove hierarchies and dimensions by clicking the links. You can also continue defining multiple hierarchies for the dimension or add new dimensions to the data object.

52.7.2 How to Create a Time Dimension

If your dimension contains a time date data type column, you can select the time levels to include in the hierarchy.

To select time levels:

1. In a dimension containing a time date data type column, add a hierarchy.

2. Select the time date data type column. If you are editing existing time levels, click **Edit Time Levels**.

   The Time Levels Definition dialog opens.

3. Click the levels to include in the hierarchy. The levels include:

   - **Year.** Year in a four digit number.
   - **Quarter.** Quarter of four quarters starting with quarter one representing January, February, March.

- **Month.** Months one through 12, starting with January.
- **Week of the Year.** Numbers for each week starting with January 1st.
- **Day of the Year.** Numbers for each day of the year starting with January 1st.
- **Day of the Month.** Numbers for each day of the month.
- **Day of the Week.** Numbers for each day of the week, starting from Sunday to Saturday.
- **Hour.** Numbers from one to twenty four.
- **Minute.** Numbers from one to 60.
- **Second.** Numbers from one to 60.

4. Click **OK** to close the dialog.

52.8 Renaming and Moving Data Objects

You can rename and move a data object without editing or clearing the data object. If you only want to change the data object name or description, use the Rename option.

52.8.1 How to Rename a Data Object

To rename a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to rename.

   The general information for the data object is displayed in the right frame.
3. Select **Rename/Move**.
4. Enter the new name, tip text, and description for the data object.
5. Click **Save changes**.

52.8.2 How to Move a Data Object

To move a data object:

1. Select **Data Objects** from the list.
2. Select the data object to rename.

   The general information for the data object is displayed in the right frame.
3. Select **Rename/Move**.
4. Click **Browse** to enter the new location for the data object.
5. Click **Save changes**.

52.9 Creating Indexes

Indexes improve performance for large data objects containing many rows. Without any indexes, accessing data requires scanning all rows in a data object. Scans are inefficient for very large data objects. Indexes can help find rows with a specified value in a column.

If the data object has an index for the columns requested, the information is found without having to look at all the data. Indexes are most useful for locating rows by values in columns, aggregating data, and sorting data.

52.9.1 How to Create an Index

You can add indexes to data objects by selecting columns to be indexed as you are creating a data object.

To add an index:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to add an index to.
3. Select **Indexes**.
4. Click **Add Index**.

   The Add Index dialog opens.
5. Enter a **Name** and **Description** for the index
6. Add as many columns as needed to create an index for the table.

   Click a column in the list on the right to remove the column from the index.
7. Click **OK**.

   The index is added and is named after the columns it contains. You can create multiple indexes. To remove an index you created, click **Remove Index** next to the Index name.

52.10 Clearing Data Objects

Clearing a data object removes the current contents without deleting the data object from the Oracle BAM ADC.

52.10.1 How to Clear a Data Object

To clear a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to clear.

   The general information for the data object is displayed in the right frame.
3. Click **Clear**.

52.11 Deleting Data Objects

When deleting data objects, you must remove referrals to the data object from reports and alerts that are using it. If the data object is in use by an active alert or report, it cannot be deleted in Oracle BAM Architect.

52.11.1 How to Delete a Data Object

To delete a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.

2. Click the data object to delete.

   The general information for the data object is displayed in the right frame.

3. Click **Delete**.

53

Creating Oracle BAM Enterprise Message Sources

This chapter contain the information required to create Enterprise Message Sources (EMS) in the Oracle BAM Architect application.

This chapter includes the following sections:

53.1 Introduction to Enterprise Message Sources

Enterprise Message Sources (EMS) are used by applications to provide direct Java Message Service (JMS) connectivity to the Oracle BAM Server. JMS is the standard messaging API for passing data between application components and allowing business integration in heterogeneous and legacy environments.

The EMS does not configure Extract Transform and Load (ETL) scenarios, but rather maps from a message directly to a data object on the Oracle BAM Server; however, you can still use XML Stylesheet Language (XSL) to perform a transformation in between. Each EMS connects to a specific JMS topic or queue, and the information is delivered into a data object in the Oracle BAM Active Data Cache. The Oracle BAM Architect web application is used to configure EMS definitions.

The following JMS providers are supported:

- Messaging for Oracle WebLogic Server
- Non-Oracle certified JMS providers:
  - IBM WebSphere MQ 6.0
  - Tibco JMS
  - Apache ActiveMQ

  See Section 53.4, "Using Foreign JMS Providers" for more information.

The following message types are supported:

- Map message

- Text message with XML payload

The following XML formatting options are supported for Text message transformation:

- Pre-processing
- Message specification
- Column value (Column values can be provided either as elements or attributes in the XML payload.)

To view the existing EMS definitions, select **Enterprise Message Sources** from the Oracle BAM Architect function list.

Figure 53–1 Oracle BAM Architect Function List

53.2 Creating Enterprise Message Sources

When you define an EMS, you specify all of the fields in the messages to be received. Some messaging systems have a variable number of user-defined fields, while other systems have a fixed number of fields.

For any string type field, you can apply formatting to that field to break apart the contents of the field into separate, individual fields. This is useful for messaging systems where you cannot create user-defined fields and the entire message body is received as one large field. The formatting specifications allow you to specify the path to a location in the XML tree, and then extract the attributes or tags as fields.

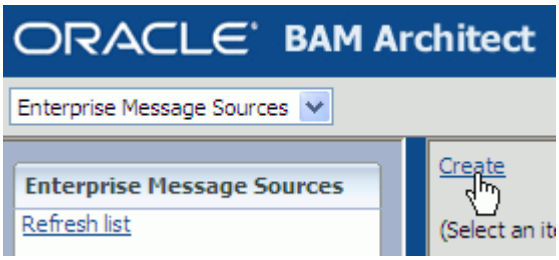
Before defining an EMS, you must be familiar with the third party application providing the messages so that you can specify the message source connection details in Oracle BAM Architect.

Furthermore, note that the JMS server (where you host queues/topics) can be configured on a different system than that which hosts the Oracle BAM Server. (For Oracle Advanced Queuing (AQ) it is acceptable to host on the same server as Oracle BAM because the database hosts the JMS server, but for other cases it is recommended to host the JMS server on another system).

53.2.1 How to Create an Enterprise Message Source

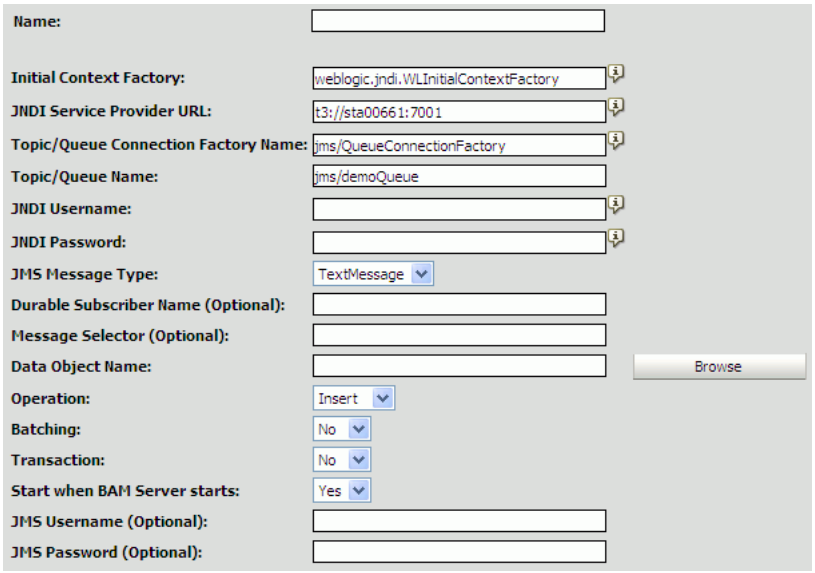
To define an EMS:

1. Select **Enterprise Message Sources** from the Oracle BAM Architect function list (see Figure 53–1).
2. Click **Create**.

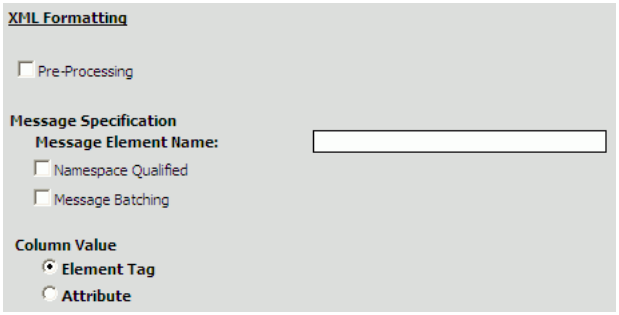


3. Using Table 53–1 as a guide, enter the appropriate values in each of the fields. Examples given are for connecting to Messaging for Oracle WebLogic Server.

> **Caution:** A single or double quotation mark in an Oracle BAM object name, such as a data object, report, or EMS name, causes a runtime error. Do not include single or double quotation marks in an Oracle BAM object name.
>
> Do not configure two Enterprise Message Sources on the same topic or queue. If you require two Enterprise Message Sources on the same Queue, each EMS must have different Message Selector value specified; otherwise, the messages are duplicated on both of the Enterprise Message Sources.
>
> If a non-Oracle WebLogic Server JMS server is used (such as Tibco) then the durable subscription name should not be repeated in any of the Enterprise Message Sources created. Some JMS servers do not allow the clients to have multiple ConnectionFactory for a single topic destination, and Oracle BAM does not support the ability to reuse the same ConnectionFactory for the same topic.



4. If you are using **TextMessage** type, configure the appropriate parameters in the XML Formatting sections, using Table 53–2 as a guide.



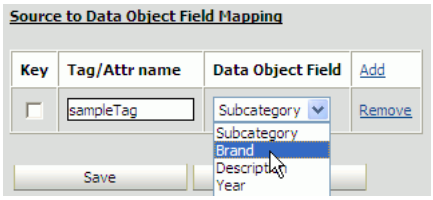
5. To configure the **DateTime Specification** in the **Source Value Formatting** section, see Section 53.2.2, "How to Configure DateTime Specification."

   Note that when **DateTime Specification** is disabled (not checked), the incoming value must be in `xsd:dateFormat`. That is, `xsd:dateFormat` ([-]CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm]) is the default format when **DateTime Specification** is not configured.

   Valid value patterns for `xsd:dateTime` include:

   - 2001-10-26T21:32:52
   - 2001-10-26T21:32:52+02:00
   - 2001-10-26T19:32:52Z
   - 2001-10-26T19:32:52+00:00
   - -2001-10-26T21:32:52
   - 2001-10-26T21:32:52.12679

6. Map fields from the source message to the selected data object in the **Source to Data Object Field Mapping** section.



   a. Click **Add** to add a mapped field.

   b. Select the **Key** checkbox if the field is a key.

   c. Enter the source tag or attribute name in the **Tag/Attr name** field.

   d. Select the target data object field from the **Data Object Field** list.

Note: When a timestamp field is included in an EMS payload, the following must be considered:

Inserting null in a timestamp field might not be in the control of a client like EMS.

When no value is given for a timestamp field (as in Oracle BAM Architect), EMS assigns the current datetime.

When the incoming timestamp value does not adhere to the `xsd:dateTime` or the datetime format specified in the EMS, the current datetime is inserted.

7. Click **Save** to save the EMS.

Table 53–1 EMS Configuration Parameters

Parameter	Description
Name	A unique display name that appears in the EMS list in Oracle BAM Architect.
Initial Context Factory	The initial context factory to be used for looking up specified JMS connection factory or destination. For example: weblogic.jndi.WLInitialContextFactory
JNDI Service Provider URL	Configuration information for the service provider to use. Used to set javax.naming.Context.PROVIDER_URL property and passed as an argument to initialContext(). An incorrect provider URL is the most common cause of errors. For example: t3://localhost:7001
Topic/Queue ConnectionFactory Name	The name used in a JNDI lookup of a previously created JMS connection factory. For example: jms/QueueConnectionFactory
Topic/Queue Name	The name used in the JNDI lookup of a previously created JMS topic or queue. For example: jms/demoQueue jms/demoTopic
JNDI Username	The identity of the principal for authenticating the JNDI service caller. This user must have RMI login permissions. Used to set javax.naming.Context.SECURITY_PRINCIPAL and passed to initialContext().
JNDI Password	The identity of the principal for authenticating the JNDI service caller. Used to set javax.naming.Context.SECURITY_CREDENTIALS and passed to initialContext().
JMS Message Type	TextMessage or MapMessage. If TextMessage is selected, XML is used to specify the contents of the payload, and an additional set of XML Formatting configuration parameters must be completed. See Table 53–2 for more information.

Table 53–1 (Cont.) EMS Configuration Parameters

Parameter	Description
Durable Subscriber Name	Enter the name of the subscriber, for example, BAMFilteredSubscription. The Durable Subscriber Name should match the event-publisher subscriber name property if it is provided. A durable subscription can preserve messages published on a topic while the subscriber is not active. It enables Oracle BAM to be disconnected from the JMS provider for periods of time, and then reconnect to the provider and process messages that were published during the disconnected period. See Section 53.3.3, "How to Subscribe and Unsubscribe Enterprise Message Sources" for information about unsubscribing an EMS from a durable subscription once the EMS is started.
Message Selector (Optional)	A single name-value pair (currently only one name-value pair is supported) that allows an application to have a JMS provider select, or filter, messages on its behalf using application-specific criteria. When this parameter is set, the application-defined message property value must match the specified criteria for it to receive messages. To set message property values, use stringProperty() method on the Message interface. he name value pair format should be *name*=*value*, for example, `message=mymessage`. The equals sign (=) is the name-value pair separator.
Data Object Name	Data object in Oracle BAM in which to deposit message data. Operations can be performed on only one data object per EMS. The data object can have Lookup columns. Click **Browse** to choose a data object.
Operation	Select the operation from the list: **Insert** inserts all new data as new rows **Upsert** merges data into existing rows **Update** updates existing rows **Delete** removes rows from the data object
Batching	Specify whether the EMS communicates with the Oracle BAM Active Data Cache API with batching enabled. Batching allows multiple messages to be inserted using a single Text Message. If Batching is disabled (the default state), each row read from JMS would be sent to the Active Data Cache as a separate unit and not part of a batch of rows. Batching properties are contained in configuration files. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.
Transaction	Enabling Transaction ensures that the operation is atomic when Batching is enabled (Batching allows multiple messages to be inserted using a single Text Message). Transaction itself does not have any impact on Active Data Cache batching, but setting Transaction to true ensures that all of the messages in Messaging Batching (when many messages are batched in a single batch) are part of an atomic operation. See Message Batching inTable 53–2.
Start when BAM Server starts	Specify whether the EMS starts reading messages and sending them to the Active Data Cache as soon as the Oracle BAM Server starts (or restarts).

Table 53–1 (Cont.) EMS Configuration Parameters

Parameter	Description
JMS Username (Optional) JMS Password (Optional)	You can optionally provide this information when a new JMS connection is created by a connection factory. Used to authenticate a connection to a JMS provider for either application-managed or container-managed authentication.

Table 53–2 EMS XML Formatting Configuration Parameters

Parameter	Description
Pre-Processing	XSL transformation can be applied to an incoming Text Message before message retrieval and column mapping are done. See Section 53.2.3, "How to Use Advanced XML Formatting" for more information. XML names can be qualified. If qualified, select the **Namespace Qualified** box and enter the namespace URI in the field.
Message Element Name	The parent element that contains column values in either its sub-elements or attributes. XML names can be qualified. If qualified, select the **Namespace Qualified** box and enter the namespace URI in the field.
Message Batching	Multiple messages can be batched in a single JMS message. If this is the case, a wrapper element must be specified as the containing element in **Batch Element Name**. If qualified, select the **Namespace Qualified** box and enter the namespace URI in the field.
Column Value	Column values can be provided using either elements or attributes in an XML payload. Specify which column value type is provided in the payload.

53.2.2 How to Configure DateTime Specification

To configure DateTime Specification:

1. Select the DateTime Specification checkbox as shown in Figure 53–2.
2. Enter the date and time pattern in the **Pattern** field.

   You can select one of the suggested supported patterns provided in the dropdown list, or enter it manually into the text box.

   You must supply a valid date and time pattern that adheres to the Java SimpleDateFormat. Table 53–3 provides the syntax elements for SimpleDateFormat, and Table 53–4 provides some examples.

Note: If you are sending datetime/timestamp data to an Oracle BAM EMS through Oracle AQ JMS, the following must be considered while configuring DateTime Specification:

The default datetime formats in Oracle database are specified either explicitly, with the NLS session parameters `NLS_DATE_FORMAT`, `NLS_TIMESTAMP_FORMAT`, and `NLS_TIMESTAMP_TZ_FORMAT`, or implicitly, with the NLS session parameter `NLS_TERRITORY`.

If trigger processing code (PL/SQL) does not override the date format with explicit formatting, the dates are formatted according to the format specified by NLS parameters for the database session, and sent accordingly to EMS. This means that the EMS DateTime Specification must have the equivalent format of NLS parameters to parse and interpret the incoming data.

However, problems arise on the EMS side if a database administrator changes the NLS parameters. It is always safe to use explicit formatting using the `to_char()` function, rather than rely on the default NLS parameters specified format.

A line such as this example in Trigger processing code

```
  '<HIREDATE>' || :new.HIREDATE || '</HIREDATE>' ||
```

should be changed to something similar to

```
  '<HIREDATE>' || to_char(:new.HIREDATE, 'MM/dd/yy HH24:MI:SS') ||
'</HIREDATE>' ||
```

The corresponding format selected from EMS DateTime Specification drop down is `MM/dd/yy H:mm:ss`.

Similarly, for timestamp data, if the format selected on the database side with the `to_char()` function is `MM/dd/yy HH24:MI:SS.FF`, the corresponding EMS DateTime Specification format is `MM/dd/yy H:mm:ss:SSS`.

Note: When you explicitly select the `HH:mm:ss` datetime format, the default value `1/1/1970` is inserted for the date, EMS ignores the date value.

When you explicitly select only the date (excluding the hour, minute, and second) as the datetime format, then the date is inserted with its hour, minute, and second set to `12:00:00 AM`. EMS ignores the time value.

3. Optionally, you can enter the locale information in the **Language**, **Country**, and **Variant** fields.

Figure 53–2 EMS Configuration Source Value Formatting Section

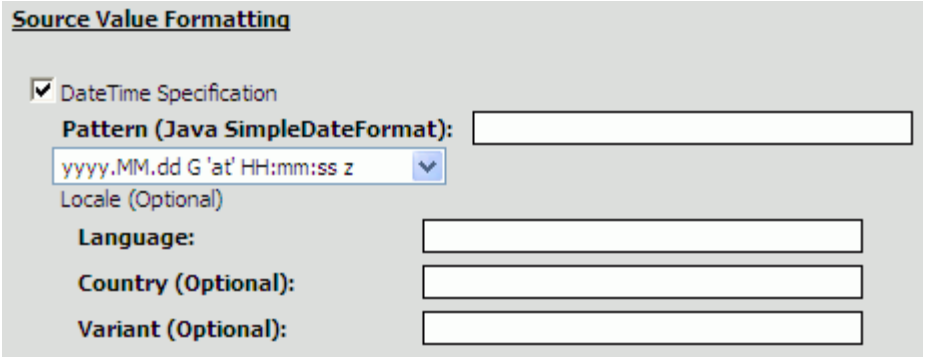


Table 53–3 Syntax Elements for SimpleDateFormat

Symbol	Meaning	Presentation	Example
G	Era	Text	AD
y	Year	Number	2003
M	Month	Text or Number	July; Jul; 07
w	Week in year (1-53)	Number	27
W	Week in month (1-5)	Number	2
D	Day in year (1-365 or 1-364)	Number	189
d	Day in a month	Number	10
F	Day of week in month (1-5)	Number	2
E	Day in week	Text	Tuesday; Tue
a	AM/PM marker	Text	AM
H	Hour (0-23)	Number	0
k	Hour (1-24)	Number	24
K	Hour (0-11 AM/PM)	Number	0
h	Hour (1-12 AM/PM)	Number	12
m	Minute in an hour	Number	30
s	Second in a minute	Number	55
S	Millisecond (0-999)	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800
’	Escape for text	Delimiter	MMM ’’01 -> Jul ’01

The examples in Table 53–4 show how date and time patterns are interpreted in the United States locale. The date and time used in all of the examples are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

Table 53–4 Date and Time Pattern Examples

Date and Time Pattern	Result
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.07.04 AD at 12:08:56 PDT

Table 53–4 (Cont.) Date and Time Pattern Examples

Date and Time Pattern	Result
"EEE, MMM d, '' yy"	Wed, Jul 4, '01
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2001-07-04T12:08:56.235-0700

53.2.3 How to Use Advanced XML Formatting

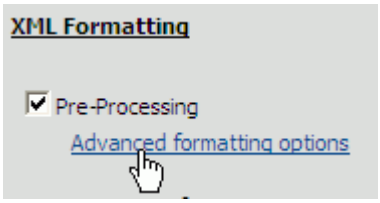
The Advanced formatting options allow an EMS to contain a user-supplied XSL Transformation (XSLT) for each formatted field in the message.

Uses for XSLT include:

- Handling of hierarchical data. The Data Flow does not handle hierarchical data. The XSLT can flatten the received XML into a single record with repeating fields.
- Handling of message queues that contain messages of multiple types in a single queue. The Data Flow requires that all records from the Message Receiver be of the same schema. The EMS output can be defined as a combined superset of the message schemas that are received, and the XSL transformation can identify each message type and map it to the superset schema as appropriate.
- Handling of XML that, while not expressing hierarchical data, does contain needed data at multiple levels in the XML. EMS formatting can only read from one level with the XML. The XSL transformation can identify the data needed at various levels in the input XML and output it all in new XML that contains all of the data combined at one level.

To specify an XSL transformation:

1. In an EMS that you are defining or editing, select **Pre-Processing** in the XML Formatting section.



2. Click **Advanced formatting options**.

   The Advanced Formatting dialog opens.

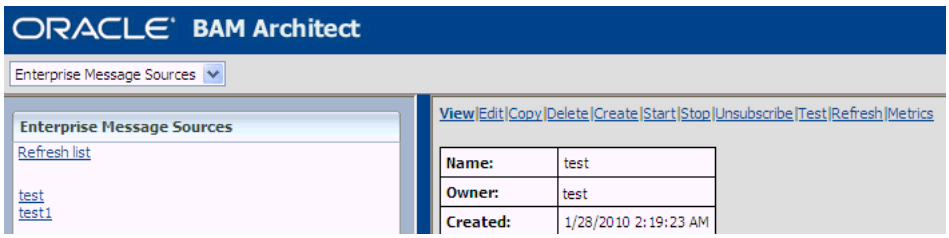
3. Type or paste the XSL markup for the transformation for the XML in this field. You might want to write the XSL markup in another editing tool and then copy and paste the code into this dialog.
4. In the **Sample XML to transform** field, type sample XML to test the transformation against. The sample XML is not saved in this dialog and is not displayed if you close and open this dialog.

5. Click **Verify transformation syntax** to validate the XSL syntax.
6. Click **Test transformation on sample XML** to test your transformation.

   The results are displayed in the field underneath the links. If any errors are found in the XSL syntax, the sample XML syntax, or during the transformation, the error text is shown in this field.

53.3 Using Enterprise Message Sources

The Enterprise Message Sources page in Oracle BAM Architect is used to view the EMS definition, and perform operations on it. Select any EMS in the **Enterprise Message Sources** list to display information about it and work with it.



Use the links displayed at the top of the EMS definition page (the pane on the right side of the browser window) to perform operations on the EMS.

The following topics describe the available operations:

- Section 53.3.1, "How to Edit, Copy, and Delete Enterprise Message Sources"
- Section 53.3.2, "How to Start and Stop Enterprise Message Sources"
- Section 53.3.3, "How to Subscribe and Unsubscribe Enterprise Message Sources"
- Section 53.3.4, "How to Test Enterprise Message Sources"
- Section 53.3.5, "How to Refresh Enterprise Message Sources"
- Section 53.3.6, "How to Monitor Enterprise Message Source Metrics"

53.3.1 How to Edit, Copy, and Delete Enterprise Message Sources

Use the **Edit**, **Copy**, and **Delete** links on an individual EMS definition page to edit, copy, or delete the current EMS definition.

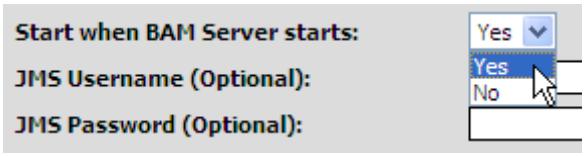
53.3.2 How to Start and Stop Enterprise Message Sources

Use the **Start** and **Stop** links on an individual EMS definition page to start and stop the EMS, which makes the consumer inactive in **Stopped** status.

For a durable subscribed EMS, clicking on **Stop** only makes the consumer inactive. It does not unsubscribe the EMS from a durable subscription. See Section 53.3.3, "How to Subscribe and Unsubscribe Enterprise Message Sources" for more information.

By default the EMS starts when the Oracle BAM Server is started.

Click **Edit** to change the **Start when BAM Server starts** property.



53.3.3 How to Subscribe and Unsubscribe Enterprise Message Sources

Use the **Unsubscribe** link on an individual EMS definition page to unsubscribe a durable subscribed EMS.

For a durable subscribed EMS, clicking on **Stop** only makes the consumer inactive with **Stopped** status.

Clicking on **Unsubscribe** unsubscribes it and the EMS status displays as **Unsubscribed**.

For non-durable subscribed EMS, clicking **Unsubscribe** does not have any effect. A message is displayed that the feature is not applicable in this case.

See Table 53–1 for information about configuring the **Durable Subscriber Name** property.

53.3.4 How to Test Enterprise Message Sources

Use the **Test** link on an individual EMS definition page to test the EMS definition against the data source and the mapped data object fields. The test results appear in the Status field in the EMS definition.

The status is reflected in the **Status** field as `Test OK` if the test is done successfully, or `Test failed - exception` are displayed when there is a problem. Also, when the **Test** link is clicked:

- If the EMS is started already, then it stops it and starts it again.
- If the EMS is in a stopped state, then it starts and stops again.

53.3.5 How to Refresh Enterprise Message Sources

Use the **Refresh** link on an individual EMS definition page to refresh the EMS definition page. Typically a user refreshes the page to obtain the current status of the EMS.

53.3.6 How to Monitor Enterprise Message Source Metrics

Use the **Metrics** link on an individual EMS definition page to monitor selected EMS statistics. The Metrics page displays the **Total Messages Received**, **Total Messages committed in ADC**, and **Total Messages Lost** counters. These values are accumulated since last start or reset.

Total Messages Lost is calculated by subtracting Total Messages committed in ADC from Total Messages Received.

Click **Refresh** to see these latest counter values.

Click **Reset** to set counter values to zero.

53.4 Using Foreign JMS Providers

Oracle WebLogic Server provides support for integrating non-Oracle WebLogic Server (foreign) JMS providers with applications deployed in it, such as Oracle BAM. Foreign JMS providers have their own JMS client and Java Naming and Directory Interface (JNDI) Client APIs. some configuration must be done to identify these depedencies and make these APIs available on Oracle WebLogic Server so that JMS resources hosted on a remote provider can be looked up by application deployed in Oracle WebLogic Server.

See "Configuring Foreign Server Resources to Access Third-Party JMS Providers" in *Oracle Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server* for more information.

Section 53.5.3, "Creating a Foreign JMS Server" in the "Use Case: Creating an EMS Against Oracle Streams AQ JMS Provider" provides a detailed example.

The high level configuration steps are:

1. Make the JMS and JNDI client library of the foreign the provider available to applications deployed on Oracle WebLogic Server.

   Identify the JMS and JNDI client Java Archive (JAR) files of the foreign provider and place them in the *DOMAIN_HOME*`/lib` directory.

2. Create a foreign server using Oracle WebLogic Server Administration Console.

   Go to **JMS Modules** in Oracle WebLogic Server Administration Console, and create a new module.

   Inside this module, click **New**, select **Foreign Server**, and create a new foreign server by navigating through all of the pages.

   Provide appropriate JNDI properties for the remote provider for the foreign server definition.

3. Create JMS resources (that is, connection factories and destinations) for the foreign JMS server.

   Inside the **Foreign Server** link, select the **Destination** tab and create links for

   - Remote ConnectionFactory
   - Remote Destination (Queue/Topic)

   Local JNDI names configured for these destinations must be used while configuring EMS to consume messages from these destinations.

4. Configure an EMS definition in Oracle BAM Architect to consume messages from foreign destinations.

   The whole process of accessing JMS resources hosted on foreign providers is transparent to Oracle BAM Server. After the previous steps have been followed correctly, remote destinations from foreign JMS providers are published on the local WL server JNDI tree, so that applications deployed on the server (like Oracle BAM EMS) can look them up, just like any other collocated Oracle WebLogic Server JMS resource. Oracle WebLogic Server takes care of communicating with the appropriate foreign JMS provider at runtime.

53.5 Use Case: Creating an EMS Against Oracle Streams AQ JMS Provider

The following are the steps to configure Oracle Streams AQ JMS Provider (AQ-JMS) in Oracle WebLogic Server, and an EMS definition in Oracle BAM Architect.

1. Creating a JMS Topic in AQ-JMS.
2. Creating a Data Source in Oracle WebLogic Server.
3. Creating a Foreign JMS Server.
4. Defining an EMS in Oracle BAM Architect.
5. Inserting and Updating Records in the SQL Table.

53.5.1 Creating a JMS Topic in AQ-JMS

Open a SQLplus command prompt and do the following:

1. Login as sysdba

   ```
   sqlplus sys as sysdba
   ```

2. Enter the password for the system dba account when prompted.
3. Create and execute the following scripts in the following order (see Example 53–1, Example 53–2, and Example 53–3 for the contents of the scripts).

   ```
   @<SCRIPT_PATH>/usertabletopiccreation.sql
   @<SCRIPT_PATH>/createtable.sql
   @<SCRIPT_PATH>/createtrigger.sql
   ```

   The scripts do the following things:

   a. Creates a fresh schema under user `MyChannelDemoUser`.
   b. Creates a JMS a topic in AQ-JMS.
   c. Creates a SQL table by name `EMP`.
   d. Creates a trigger that publishes messages to AQ-JMS topic on inset/update on `EMP`.

Example 53–1 Contents of usertabletopiccreation.sql

```
DROP USER MyChannelDemoUser CASCADE;

GRANT connect, resource,AQ_ADMINISTRATOR_ROLE TO MyChannelDemoUser IDENTIFIED BY
 MyChannelDemoPassword;
GRANT execute ON sys.dbms_aqadm TO MyChannelDemoUser;
GRANT execute ON sys.dbms_aq    TO MyChannelDemoUser;
GRANT execute ON sys.dbms_aqin  TO MyChannelDemoUser;
GRANT execute ON sys.dbms_aqjms TO MyChannelDemoUser;

connect MyChannelDemoUser/MyChannelDemoPassword;

BEGIN
--dbms_aqadm.stop_queue( queue_name => 'MY_TOPIC' );
--dbms_aqadm.drop_queue( queue_name  => 'MY_TOPIC');
--DBMS_AQADM.DROP_QUEUE_TABLE (queue_table => 'TTab');
dbms_aqadm.create_queue_table( queue_table => 'TTab', queue_payload_type =>
 'sys.aq$_jms_text_message', multiple_consumers => true );
dbms_aqadm.create_queue( queue_name  => 'MY_TOPIC', queue_table => 'TTab' );
```

```
dbms_aqadm.start_queue( queue_name => 'MY_TOPIC' );
END;
/
```

Example 53–2 Contents of createtable.sql

```
connect MyChannelDemoUser/MyChannelDemoPassword;

CREATE TABLE EMP ( EMPNO NUMBER(4), ENAME VARCHAR2(10),  JOB VARCHAR2(9), MGR
 NUMBER(4), HIREDATE DATE, SAL NUMBER(7,2), COMM NUMBER(7,2), DEPTNO NUMBER(2) );

quit;
```

Example 53–3 Contents of createtrigger.sql

```
connect MyChannelDemoUser/MyChannelDemoPassword;
create or replace
trigger employee AFTER INSERT OR Update ON EMP
 FOR each row
    declare
       xml_complete varchar2(1000);
       v_enqueue_options dbms_aq.enqueue_options_t;
       v_message_properties dbms_aq.message_properties_t;
       v_msgid raw(16);
       temp sys.aq$_jms_text_message;
       v_recipients        dbms_aq.aq$_recipient_list_t;

    Begin
         temp:=sys.aq$_jms_text_message.construct;
 xml_complete :=
 '<?xml version="1.0"?>' ||
 '<row>' ||
 '<EMPNO>' || :new.EMPNO || '</EMPNO>' ||
 '<ENAME>' || :new.ENAME || '</ENAME>' ||
 '<JOB>' || :new.JOB || '</JOB>' ||
 '<MGR>' || :new.MGR || '</MGR>' ||
 '<HIREDATE>' || :new.HIREDATE || '</HIREDATE>' ||
 '<SAL>' || :new.SAL || '</SAL>' ||
 '<COMM>' || :new.COMM || '</COMM>' ||
 '<DEPTNO>' || :new.DEPTNO || '</DEPTNO>' ||
 '</row>' ;
 temp.set_text(xml_complete);
      dbms_aq.enqueue(queue_name => 'MY_TOPIC',
         enqueue_options => v_enqueue_options,
         message_properties => v_message_properties,
         payload => temp,
         msgid => v_msgid );
  End ;
/
quit;
```

53.5.2 Creating a Data Source in Oracle WebLogic Server

You can skip this step if a data source exists. An existing data source can also be reused in this section.

1. Open Oracle WebLogic Server Administration Console at

   `http://hostname:7001/console`

   where *hostname* is the name of the system where Oracle BAM Server is installed.

2. After logging into the console click the **Data Sources** link in the **JDBC** section, and click **New**.
3. Enter a name for the data source (For example, `BAMAQDataSource`).
4. Enter a JNDI name from the data source (for example, `jdbc/oracle/bamaq`). This name is used to configure the foreign JMS server.
5. Select **Oracle** to be the **Database Type**.
6. Select **Oracle's Driver (Thin)** for **Database Driver** field, and click **Next**.
7. Uncheck **Support Global Transaction**, and click **Next**.
8. Enter your database SID in the **Database Name** field (for example, `ORCL`).
9. Enter the hostname of the system where the database is installed as the **Host Name** (for example, `localhost`).
10. Enter data base port number (for example, 1521).
11. Enter the user name (for example, `MyChannelDemoUser`).
12. Enter the password, and click **Next**.
13. Click ***Test Configuration*** to test the configuration.
14. After it is successful, click **Finish**.

53.5.3 Creating a Foreign JMS Server

To create a foreign JMS server:

1. Add as an Oracle WebLogic Server JMS module.
   a. In the Oracle WebLogic Server Administration Console, from the home page, go to the JMS Modules page.
   b. Click **New** to create an Oracle WebLogic Server JMS module.
   c. Enter a name for the JMS module (for example, `BAMAQsystemModule`).
   d. Click **Next** and assign appropriate targets.
   e. Click **Next**, and click **Finish**.
2. Add an AQ-JMS foreign server to the JMS module.
   a. Select the JMS module that you just created.
   b. Click **New**, and go to the list of JMS resources to add.
   c. Select the **Foreign Server** option, and click **Next**.
   d. Enter a name for the foreign server (for example, `BAMAQForeignServer`), and click **Finish**.
3. Configure the AQ-JMS foreign server.
   a. Select the AQ-JMS foreign server that you created.
   b. In the **JNDI Initial Context Factory** field, enter

      ```
      oracle.jms.AQjmsInitialContextFactory
      ```

   c. In the **JNDI Properties** area, enter

      ```
      datasource=datasource_jndi_location
      ```

where *datasource_jndi_location* is the JNDI location of your data source (for example, jdbc/oracle/bamaq).

4. Add connection factories to the AQ-JMS foreign server.
   a. Select the AQ-JMS foreign server that you created.
   b. Select the **Connection Factories** tab.
   c. Enter a name for the connection factory. This is a logical name referenced by Oracle WebLogic Server.
   d. In the **Local JNDI Name** field, enter the local JNDI name that is used by The Oracle BAM EMS to look up this connection factory (For example, jms/BAMAQTopicCF).
   e. In the **Remote JNDI Name** field, enter:

      ```
      - TopicConnectionFactory (select for this use case)
      - QueueConnectionFactory
      - ConnectionFactory
      ```

   f. Click **OK**.
5. Add destinations to the AQ-JMS foreign server.
   a. Select the AQ-JMS foreign server that you created.
   b. Select the **Destinations** tab.
   c. Enter a name for this destination. This is a logical name referenced by Oracle WebLogic Server, and it has nothing to do with the destination name.
   d. In the **Local JNDI Name** field, enter the local JNDI name that is used by the Oracle BAM EMS to lookup this destination (for example, jms/BAMAQTopic).
   e. In the **Remote JNDI Name** field, if the destination is a queue, enter the following value:

      Queues/*queue_name*

      If the destination is a topic enter the following value:

      Topics/*topic_name*

   f. Click **OK**.
6. Restart Oracle WebLogic Server.

53.5.4 Defining an EMS in Oracle BAM Architect

1. Open Oracle BAM Architect, and select **Enterprise Message Sources** in the dropdown list.
2. Enter the message source information you just created.
3. Enter the **Initial Context Factory** value:

   weblogic.jndi.WLInitialContextFactory

4. Enter the JNDI provider URL:

   t3://*hostname*:7001

5. Enter the **Connection Factory Name** (for example, jms/BAMAQTopicCF).
6. Enter the **Destination Name** (for example, jms/BAMAQTopic).

7. Choose the Oracle BAM data object to send the values received from AQ-JMS server.
8. Complete the source-to-data object field mapping so that data from the incoming XML can be mapped to an appropriate field in selected data object.

53.5.5 Inserting and Updating Records in the SQL Table

Now you can test the functionality end to end by inserting or updating some records in the EMP database table.

You can use SQLPlus to run SQL queries.

Now you should see the values from the record being inserted into data object.

For example,

```
insert into emp values (25,'Ford','ANALYST',7566,sysdate,60000,3000,20);

update emp set ENAME='McOwen' where ENAME='Ford';
```

54

Using Oracle Data Integrator With Oracle BAM

This chapter provides information about the Oracle Data Integrator integration with Oracle Business Activity Monitoring.

This chapter includes the following sections:

Oracle Data Integrator documentation is located on the Oracle Technology Network web site at the following location:

`http://www.oracle.com/technetwork/middleware/data-integrator`

54.1 Introduction to Using the Oracle Data Integrator With Oracle Business Activity Monitoring

This document assumes the following:

- The Oracle database is installed and you can connect to it.
- Oracle BAM is installed and running.
- Oracle Data Integrator installed and the basic configuration is done (the Oracle Data Integrator Master repository is created, repository connections are configured, Work repositories are created and connected, and any source topologies are configured).
- If Oracle Data Integrator is installed on a separate host, Java 1.6 must be installed on the Oracle Data Integrator host before you can work with the Oracle BAM and Oracle Data Integrator integration.

When using Oracle Data Integrator with Oracle BAM, keep the following in mind:

- Within the Oracle Data Integrator interface you must add quotation marks around field names that contain spaces.
- Oracle Data Integrator cannot insert data into Oracle BAM read-only fields of type Lookup, Calculated, Auto-incrementing integer, and Timestamp. These fields are automatically populated. Although Oracle Data Integrator enables you to select these fields as target fields, running Oracle Data Integrator with these fields populated throws an exception.
- Do not use Oracle BAM as a staging area (for example, if Oracle BAM is used as a source (as when using a loading knowledge module), do not use this source as staging area, and if Oracle BAM is being used as a target (as when using an integration knowledge module) do not use that target as staging area.

54.2 Installing the Oracle Data Integrator Integration Files

There are two ways to set up the Oracle BAM and Oracle Data Integrator integration.

The first method uses an installation script, typically when Oracle Data Integrator and Oracle BAM are deployed on the same system or the same network file system (Section 54.2.1, "How to Install Integration Files Using the Script").

The second method uses manual steps to configure the properties and copy the required files to the Oracle Data Integrator directories (Section 54.2.2, "How to Manually Install Integration Files"). This method is typically used if you are unable to map the *ODI_HOME* drive from the system where Oracle BAM is installed (usually when Oracle Data Integrator and Oracle BAM are installed in different network or file system).

The logs contain information about the installation and the integration messages. See Section 54.2.3, "Using the Logs" for more information.

Recommended Memory Settings for Using Oracle Data Integrator with Oracle BAM

The default memory settings for Oracle Data Integrator are included in the `odiparams.sh` script (or `odiparams.bat` for windows). The values for the `ODI_INIT_HEAP` and `ODI_MAX_HEAP` properties default to 32M and 256M. It is recommended that you change these values to 256 M and 1024 M respectively. This enhances Oracle Data Integrator performance. Otherwise, Oracle Data Integrator `OutOfMemory` errors may occur, especially when running memory intensive tasks.

54.2.1 How to Install Integration Files Using the Script

Use the installation script when you have Oracle Data Integrator and Oracle BAM installed on the same system or the same network file system.

A log file called `utility.log` is created if there is a problem with the installation. The file location is controlled by the `utility.logging.properties` file. See Section 54.2.3, "Using the Logs" for more information.

To install the integration files:

1. Verify that Oracle BAM Server is running and reachable from the Oracle Data Integrator host.
2. On the Oracle BAM host, go to the *ORACLE_HOME*`\bam\config` directory and edit the `bam_odi_configuration.properties` file.

- **ODI_HOME**

  This property identifies the path to the Oracle Data Integrator home directory.

  The default value on Linux is `/scratch/$user/ODI_HOME/oracledi`.

  On Microsoft Windows systems, use the short 8-character name convention. Also, use double back-slashes (\\) to denote a directory separator. For example, `C:\Program Files\ODI_HOME\oracledi` would appear as:

  ```
  ODI_HOME = C:\\Progra~1\\ODI_HOME\\oracledi
  ```

  Note: If Oracle BAM Server and Oracle Data Integrator are deployed on two different hosts, then you must map the Oracle Data Integrator drive on the Oracle BAM system, and then set the *ODI_HOME* path using that mapped drive to successfully make use of the integration configuration scripts. If drive mapping is not possible see Section 54.2.2, "How to Manually Install Integration Files."

- **WL_SERVER**

  This property identifies the Oracle WebLogic Server folder name on the Oracle BAM system.

  The default value is `wlserver_10.3`.

3. Execute `bam_odi_configuration.sh` (or `bam_odi_configuration.bat` on a Microsoft Windows host) in *ORACLE_HOME*`\bam\bin`.

   Note: The configuration script does the following steps:

   1. Recreates the Oracle Data Integrator EDS enterprise data sources.
   2. Modifies the `odiparams.sh` file.
   3. Copies the Oracle BAM JAR files and knowledge modules.

   In an existing working Oracle Data Integrator environment, you need only to copy the Oracle BAM artifacts. To do this run the script with the `PATCH` command line parameter. When the script finds the command line parameter it performs only step 3 above and skip the other steps. For example:

   ```
   sh bam_odi_configuration.sh PATCH
   ```

   Enter the values as prompted by the script. You must have the Oracle Data Integrator Master and Oracle Data Integrator Work repository account credentials to complete the script execution.

   Note that the prompts displayed with [*value*] have default values in the brackets. Press **Enter** to choose the default. If there is no bracketed default value displayed, an input value is required, or the script stops.

   The script creates the resources required in the Oracle BAM web applications, sets the Oracle BAM configuration properties in Oracle Data Integrator, generates a Oracle WebLogic Server client Java Archive (JAR) to deploy to the Oracle Data Integrator system, and copies all of the required files into the appropriate Oracle Data Integrator directories.

While the script is running the following message may appear: "Trying to contact Oracle BAM Server. It may take few minutes." If Oracle BAM Server cannot be reached, the script retries the connection multiple times.

Note: If you cannot use the script in your environment, use the instructions in Section 54.2.2, "How to Manually Install Integration Files."

Note: Every time `bam_odi_configuration.sh` is run, a backup of the `BAMODIConfig.xml` file is created in the same directory with a time stamp, and the old file is overwritten with the new file. If you made any changes to the property settings in the old version of `BAMODIConfig.xml`, those changes must be made again in the latest version.

Now you can create an Oracle BAM target in the Oracle Data Integrator Topology Manager. See Section 54.4, "Creating the Oracle BAM Target" for instructions.

54.2.2 How to Manually Install Integration Files

Use these steps if Oracle Data Integrator and Oracle BAM Server are installed on hosts in different networks, or for any reason you cannot use the script in your environment.

There are four major steps to this process:

1. Set JAVA_HOME
2. Create External Data Sources for Oracle Data Integrator
3. Set Oracle Data Integrator Configuration Properties
4. Copy files to Oracle Data Integrator Directories
5. Generate the Oracle WebLogic Server Client JAR

Set JAVA_HOME

The environment variable `JAVA_HOME` must be set to Java version 1.6.x in the environment in which an Oracle Data Integrator application is invoked. This means that Java version 1.6.x must be installed on the host. To set the environment variable:

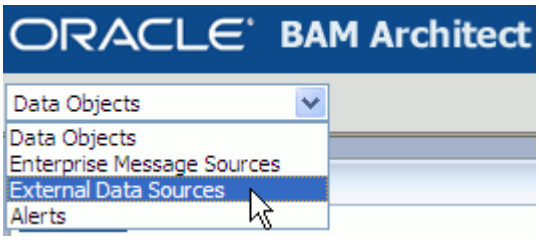
On Microsoft Windows, go to the Control Panel, click the System icon. In the System Properties, go to the Advanced tab, and then click the Environment Variables button. In the Environment Variables window, create or modify a variable named `JAVA_HOME` for the user (upper box), and set the value to the path for the Java installation (for example: `c:\PROGRA~1\Java\jdk1.6.0_12`). Click OK. When you launch Oracle Data Integrator, be sure to do it from a fresh command prompt, to pick up the new environment variable.

On UNIX, follow the procedure for the shell script to create the environment variable `JAVA_HOME`. This can be done in a startup script (such as .cshrc in the user's home directory) or on the command line before invoking Oracle Data Integrator.

Create External Data Sources for Oracle Data Integrator

Create the external data sources in Oracle BAM Architect.

1. Open Oracle BAM Architect and select the External Data Sources page.



2. Click **Create**, and configure the two external data sources (ODI_Master and ODI_Work) with the values shown in Table 54–1 and Table 54–2.

Table 54–1 ODI_Master external data source values

Property	Value
External Data Source Name	ODI_Master
Driver	oracle.jdbc.driver.OracleDriver
Login	Oracle Data Integrator Master repository account user name
Password	Oracle Data Integrator Master repository account password
Connection String	jdbc:oracle:thin:*ip_address*:*port_number*:*db_service_name*

Table 54–2 ODI_Work external data source values

Property	Value
External Data Source Name	ODI_Work
Driver	oracle.jdbc.driver.OracleDriver
Login	Oracle Data Integrator Work repository account user name
Password	Oracle Data Integrator Work repository account password
Connection String	jdbc:oracle:thin:*ip_address*:*port_number*:*db_service_name*

Set Oracle Data Integrator Configuration Properties

Modify the `ODI_JAVA_OPTIONS` and `ODI_ADDITIONAL_CLASSPATH` values in the `odiparams.sh(bat)` file located in *`ODI_HOME`*`/bin` as shown in Example 54–1 and Example 54–2.

Example 54–1 ODI_JAVA_OPTIONS Modification

On Microsoft Windows:

```
SET ODI_JAVA_OPTIONS=-Djava.security.policy=server.policy
 -Djava.util.logging.config.file=../lib/bam_odi.logging.properties
```

On Linux:

```
SET ODI_JAVA_OPTIONS="-Djava.security.policy=server.policy
 -Djava.util.logging.config.file=../lib/bam_odi.logging.properties"
```

Example 54–2 ODI_ADDITIONAL_CLASSPATH Modification

```
SET ODI_ADDITIONAL_CLASSPATH=../lib/weblogic/wlfullclient.jar
```

Copy files to Oracle Data Integrator Directories

This procedure copies several JAR files, logging properties files, and knowledge modules into the Oracle Data Integrator directories.

1. Copy the following files from *ORACLE_HOME*/bam/modules/oracle.bam_11.1.1 to *ODI_HOME*/lib:
   - oracle-bam-common.jar
   - oracle-bam-etl.jar
   - oracle-bam-adc-ejb.jar
2. Copy the following files from *ORACLE_HOME*/bam/modules/oracle.bam.thirdparty_11.1.1 to *ODI_HOME*/lib:
   - commons-codec-1.3.jar
   - xstream-1.1.3.jar
3. Copy the following file from *ORACLE_HOME*/modules/oracle.odl_11.1.1 to *ODI_HOME*/lib:
   - ojdl.jar
4. Copy the following file from *ORACLE_HOME*/modules/oracle.jps_11.1.1 to *ODI_HOME*/lib:
   - jps-api.jar
5. Copy the following file from *ORACLE_HOME*/modules/oracle.dms_11.1.1 to *ODI_HOME*/lib:
   - dms.jar
6. Copy the following file from *ORACLE_HOME*/modules to *ODI_HOME*/lib:
   - org.jaxen_1.1.1.jar
7. Copy the following file from *ORACLE_HOME*/bam/config to *ODI_HOME*/lib:
   - bam.odi.logging.properties
8. Copy the following file from *ORACLE_HOME*/bam/ODI/config to *ODI_HOME*/lib/config:
   - BAMODIConfig.xml
9. Copy all of the XML files from *ORACLE_HOME*/bam/odi/knowledge_modules to *ODI_HOME*/impexp.

Generate the Oracle WebLogic Server Client JAR

1. Generate a wlfullclient.jar file using the Oracle WebLogic Server JarBuilder tool. See "Using the WebLogic JARBuilder tool" in *Oracle Fusion Middleware Programming Stand-alone Clients for Oracle WebLogic Server* for instructions.
2. Create a subdirectory called *ODI_HOME*/oracledi/lib/weblogic.

3. Copy `wlfullclient.jar` into *ODI_HOME*`/oracledi/lib/weblogic`.

54.2.3 Using the Logs

Install Log

Part of the installation process uses Oracle BAM ICommand, and the logs associated with this process are written to files in the same directory where the configuration script is run (*ORACLE_HOME*`\bam\bin`).

The logging properties for installation logs are configured in the `utility.logging.properties` file in the same directory. The default logging level is set to `INFO`.

Runtime Log

The `bam_odi.logging.properties` file is used to configure logging for messages that occur when Oracle Data Integrator is running with Oracle BAM. This file is located in *ODI_HOME*`/lib`.

54.3 Using Oracle BAM Knowledge Modules

Knowledge modules are generic code templates containing the sequence of commands necessary for a data integration pattern. A knowledge module contains the knowledge required by Oracle Data Integrator to perform a specific set of tasks against a specific storage technology. It defines methods related to a given storage technology and it enables processes generation for that technology.

There are different knowledge modules for loading (from the source data store), integration (to target data store), checking, reverse-engineering, journalizing and creating services. All knowledge modules work by generating code to be executed at runtime by knowledge module Interpreter.

There is a set of knowledge modules specific to Oracle BAM functionality within Oracle Data Integrator. These knowledge modules are installed in the *ODI_HOME*`/oracledi/impexp` directory when the integration files are installed. To use these Oracle BAM-specific knowledge modules, you must import them into the appropriate projects in the Oracle Data Integrator Designer application. Table 54–3 describes the Oracle BAM-specific knowledge modules.

For information about importing knowledge modules, see "Importing a KM" in *Oracle Data Integrator User's Guide*. Oracle Data Integrator documentation is located on the Oracle Technology Network web site at the following location:

`http://www.oracle.com/technetwork/middleware/data-integrator`

Table 54–3 Oracle BAM Knowledge Modules

Knowledge Module	Description
CKM Get Oracle BAM Metadata	A check knowledge module that is used internally before integration knowledge module steps. This check knowledge module is the default knowledge module in Oracle BAM technology, and it is automatically acquired by Oracle Data Integrator. This check knowledge module creates two arrays which are later used by Oracle BAM-specific integration knowledge modules in the same Java session. This knowledge module has no options.

Table 54–3 (Cont.) Oracle BAM Knowledge Modules

Knowledge Module	Description
IKM SQL to Oracle BAM (delete)	An integration knowledge module that can delete rows from Oracle BAM data objects by sending matching key column values. It has the following options: COMMIT_SIZE BATCH_SIZE DATETIME_PATTERN KEY_CONDITION LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT
IKM SQL to Oracle BAM (insert)	An integration knowledge module that can insert rows to Oracle BAM data objects from heterogeneous data sources. It has the following options: BATCH_SIZE COMMIT_SIZE CREATE_TARG_TABLE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT
IKM SQL to Oracle BAM (looksert natural)	An integration knowledge module that can insert rows into Oracle BAM data objects from heterogeneous data sources. It differs from IKM SQL to Oracle BAM (insert) by also inserting new entries in dimension tables (that is, the data object to which the lookup column refers) if it does not yet exist. Looksert integration knowledge modules do an insert into an Oracle BAM target based on a lookup field. Typically, this is used to load a fact table in a star schema. (A star schema is characterized by one or more very large fact tables that contain the primary information in the data warehouse, and some much smaller dimension tables (or lookup tables), each of which contains information about the entries for a particular attribute in the fact table.) This integration knowledge module is provided for better performance. It has the following options: BATCH_SIZE COMMIT_SIZE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT NON_KEY_MATCHING

Table 54–3 (Cont.) Oracle BAM Knowledge Modules

Knowledge Module	Description
IKM SQL to Oracle BAM (looksert surrogate)	An integration knowledge module that can insert rows into Oracle BAM data objects from heterogeneous data sources. It is similar to IKM SQL to Oracle BAM (looksert natural) and differs in using a surrogate key instead of a natural key between a fact data object and dimension object. Looksert integration knowledge modules do an insert into an Oracle BAM data object based on a lookup field. Typically, this used to load a fact table in a star schema. (A star schema is characterized by one or more very large fact tables that contain the primary information in the data warehouse, and some much smaller dimension tables (or lookup tables), each of which contains information about the entries for a particular attribute in the fact table.) If the value for a lookup field does not exist in the relevant dimension table, the value is automatically inserted. This integration knowledge module must be used with LKM Get Source Metadata and CKM Get Oracle BAM Metadata. This knowledge module has the following options: BATCH_SIZE COMMIT_SIZE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT NON_KEY_MATCHING
IKM SQL to Oracle BAM (update)	An integration knowledge module that can update rows in Oracle BAM data objects from heterogeneous data sources. It has the following options: BATCH_SIZE COMMIT_SIZE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT

Table 54–3 (Cont.) Oracle BAM Knowledge Modules

Knowledge Module	Description
IKM SQL to Oracle BAM (upsert)	An integration knowledge module that can merge (upsert) rows (that is, update a data object if matching row exists or insert data object if a new row) to Oracle BAM data objects from heterogeneous data sources. It has the following options: BATCH_SIZE COMMIT_SIZE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT **Note:** During execution, the number of upsert operations are reported in the **No. of Updates** field, because the Oracle Data Integrator Operator user interface does not have a **No. of Upserts** field. Furthermore, the count for all of the inserts and updates to the Oracle BAM database are reported in the **Updates** field, and are not reported separately.
LKM Get Source Metadata	A loading knowledge module. This is not a traditional loading knowledge module because it does not load any data from the source to staging area. Instead it simply gathers the metadata that is required by the integration knowledge module IKM SQL to Oracle BAM (looksert surrogate). IKM ORACLE to BAM (looksert surrogate) performs the task of loading directly from a SQL source into the Oracle BAM target. In doing so, it uses the metadata provided by LKM Get Source Metadata. This knowledge module has no options.
LKM Oracle BAM to SQL	A loading knowledge module that allows client applications to load data from Oracle BAM. If using an Oracle BAM loading knowledge module as a source in an interface (for example LKM Oracle BAM to SQL), the user must change the default **execute on** button for each mapped field in the target to **staging area**. If left at the default **source**, erroneous results may occur. Technologies that do not allow for a staging area, such as Oracle BAM, should not have transformations performed on them. It has the following options: DELETE_TEMPORARY_OBJECTS DROP_PURGE LAST_BAM_TASK

Table 54–3 (Cont.) Oracle BAM Knowledge Modules

Knowledge Module	Description
RKM Oracle BAM	A customized reverse engineering knowledge module for Oracle BAM. It has the following options: GET_COLUMNS GET_FOREIGN_KEYS GET_INDEXES GET_PRIMARY_KEYS LOG_FILE_NAME USE_LOG

Table 54–4 describes the parameters used in Oracle BAM knowledge modules.

Table 54–4 Oracle BAM Knowledge Module Parameters

Parameter	Description
BATCH_SIZE	The maximum number of records which are sent as a batch across from the client to the server. The batch size that is used to send batches from the client to the server. As larger hosts are used with bigger Java Virtual Machine sizes, this parameter can be increased to improve performance. Default value: 1024
COMMIT_SIZE	The maximum number of records in a single transaction. The default, 0, means commit all input records in one transaction. A positive, nonzero, value denotes that the maximum number of records to be committed at a time. Negative values for this option are invalid. Default value: 0
CREATE_TARG_TABLE	Select this option to create the target data object on Oracle BAM Server.
DATETIME_PATTERN	This option and Locale specifications (for example, LOCALE_LANGUAGE, LOCALE_COUNTRY, and LOCALE_VARIANT) are used to construct a Java SimpleDateFormat object which is used in parsing the date and time data strings. See Section 53.2.2, "How to Configure DateTime Specification" for information about SimpleDateFormat.
DELETE_TEMPORARY_OBJECTS	Set this option to `NO` to retain temporary objects after integration. This option is useful for debugging.
DROP_PURGE	Set this option to `YES` to not only drop the work table, but purge it as well. When a table is dropped, it is recoverable in the database's recycle bin. When the table is dropped and purged, it is permanently deleted.
GET_COLUMNS	Set to `Yes` to reverse engineer the columns.
GET_FOREIGN_KEYS	Set to `Yes` to reverse engineer the foreign keys.
GET_INDEXES	Set to `Yes` to reverse engineer the indexes.
GET_PRIMARY_KEYS	Set to `Yes` to reverse engineer the primary keys.

Table 54–4 (Cont.) Oracle BAM Knowledge Module Parameters

Parameter	Description
KEY_CONDITION	Set this option to match one or more corresponding rows from source to target. Use the following operators: *, =, !=, <, <=, >, >=. The match value (that is, the `where` clause value) should be supplied as the mapping value for the target data store's key field in the **Diagram** tab for the interface in Oracle Data Integrator Designer. Note that when the * operator is chosen as the KEY_ CONDITION option value, all rows are deleted from the target data store, regardless of its key field's mapping value.
LAST_BAM_TASK	Use this option to manage the life cycle of the Oracle BAM JDBC connection. If this task is the last Oracle BAM task in the work flow, it closes the JDBC connection; otherwise, it leaves the connection open.
LOCALE_COUNTRY	The country option is a valid ISO Country Code. These codes are the upper-case, two-letter codes as defined by ISO-3166. This option plus `LOCALE_LANGUAGE` and `LOCALE_VARIANT` are used to construct a Java Locale object.
LOCALE_LANGUAGE	The language option is a valid ISO Language Code. These codes are the lower-case, two-letter codes as defined by ISO-639. This option plus `LOCALE_COUNTRY` and `LOCALE_VARIANT` are used to construct a Java Locale object.
LOCALE_VARIANT	The variant option is a vendor or browser-specific code. For example, use `WIN` for Windows, `MAC` for Macintosh, and `POSIX` for POSIX. Where there are two variants, separate them with an underscore, and put the most important one first. For example, a Traditional Spanish collation might construct a locale with parameters for language, country and variant as: `es`, `ES`, `Traditional_WIN`. This option plus LOCALE_LANGUAGE and LOCALE_ COUNTRY are used to construct a Java Locale object.
LOG_FILE_NAME	Specify when `USE_LOG` is set to `Yes`. Specify the path and file name of the log. Be sure to set this property value properly (that is, choose a location where user has write permissions) before running the reverse engineering.
NON_KEY_MATCHING	Determines if the incoming non-key column values are to be compared to the non-key column values in the dimension table. If `NON_KEY_MATCHING` is set to `true`, if the incoming non-key column values match those in the dimension table, the row is inserted into the fact table (which is the target data store). Otherwise, that row insert fails, which might even lead to the entire transaction being rolled back (in case `COMMIT_SIZE` was set to 0). A `COMMIT_SIZE` of 1 results in only this row being rolled back and ignored, and all other row inserts progress as usual. If `NON_KEY_MATCHING` is set to `false` and lookup succeeds, incoming non-key column values for the dimension table are ignored.
USE_LOG	Set to `Yes` if you want the reverse-engineering process log details in a log file. Specify the log file location using the `LOG_ FILE_NAME` option.

54.4 Creating the Oracle BAM Target

This section details the steps for creating an Oracle BAM target using the Oracle Data Integrator Topology Manager.

For more information about using Oracle Data Integrator, see the Oracle Data Integrator documentation located on the Oracle Technology Network web site at:

http://www.oracle.com/technetwork/middleware/data-integrator

54.4.1 How to Create the Oracle BAM Target

To create an Oracle BAM Target in Oracle Data Integrator:

1. Open the Oracle Data Integrator Topology Manager.
2. Go to **Physical Architecture > Technologies > Oracle BAM**.
3. Right-click and choose **Insert Data Server**.
4. Configure the following in the **Data Server Definition** tab:
   - **Name**: Oracle BAM target name
   - **Server (Data Server)**: leave blank
   - **User**: Oracle BAM Administrator user name
   - **Password**: Oracle BAM Administrator password
5. Configure the following in the **JDBC** tab:
   - **JDBC Driver**: any_text_will_do
   - **JDBC URL**: `instance1:`*`hostname`*`:`*`port_number`*

     The `instance1` string can be any string.

     The *`hostname`* value must be the same as the `ServerName` property value in the `BAMCommonConfig.xml` file, and the *`port_number`* value must be the same as the `ServerPort` property value in the `BAMCommonConfig.xml` file.
   - Do not use the **Test** button in this dialog, because it is not functional for the integration between Oracle BAM and Oracle Data Integrator. After you successfully reverse engineer the data objects in the Oracle BAM model, then you can verify that the connection information is correct.
6. Click **OK**.
7. Configure the following in the Physical Data Server dialog:
   - In the **Physical Schema Definition** tab:
     - Modify the **Local Object Mask** to be `%OBJECT`.
   - In the **Context** tab:
     - Create a new row which automatically introduces a row with the **Context** name `Global`.

       For that row, the **Logical Schema** value is initially `<Undefined>`. You must select the `<Undefined>` text and replace it with the display name for Oracle BAM.
     - Type in a display name for the Oracle BAM target such as `BAM_TARGET` as the name of a new **Logical Schema**. Oracle Data Integrator automatically creates the logical schema.

- Click **OK**.

54.5 Reverse Engineering the Oracle BAM Schema

You must be able to see the Oracle BAM schema in Oracle Data Integrator before you can do any operations on a particular Oracle BAM data object. To accomplish this, the Oracle BAM schema (that is, all of the data objects in Oracle BAM) must be reverse engineered using the RKM Oracle BAM knowledge module described in Table 54–3.

To reverse engineer the Oracle BAM schema:

1. Create a Model on the Oracle BAM target created in Section 54.4, "Creating the Oracle BAM Target."
2. Configure the following in the **Definition** tab:
   - **Technology:** Oracle BAM target
   - **Logical Schema:** BAM_TARGET
3. Configure the following in the **Reverse** tab:
   - Choose **Customized** reverse.
   - **Context:** Global
   - **Select your KM:** RKM Oracle BAM

   Note: Because this reverse engineering is not done using a JDBC driver, it is not possible to right-click a data store and view its data.

4. Click **Reverse** to begin reverse engineering.

   You can monitor the reverse engineering process by viewing its progress in Oracle Data Integrator Operator.

   The reverse engineering produces a `reverse.log` file. The name and location of the log file can be changed in the `LOG_FILE_NAME` option.

   Any of the knowledge module options can be changed on this tab (they are described in Table 54–4).
5. When reverse engineering is complete, the metadata for the Oracle BAM schema appears in Oracle Data Integrator Designer, under the Oracle BAM target node.

54.6 Updating the Oracle Data Integrator External Data Source Definition

When you install the Oracle BAM integration files for Oracle Data Integrator with a correctly populated properties file, you are not required to do any other configuration in Oracle BAM. Two external data source (EDS) definitions are created during the installation process, and they are populated with the correct values to connect Oracle BAM Server with the ODI_Master and ODI_Work repositories in Oracle Data Integrator. These Oracle Data Integrator-specific EDS definitions must never be deleted.

There are cases in which you must update the Oracle Data Integrator EDS definitions:

- If you change the Oracle Data Integrator login credentials, you must update the Oracle Data Integrator EDS definitions in Oracle BAM Architect.

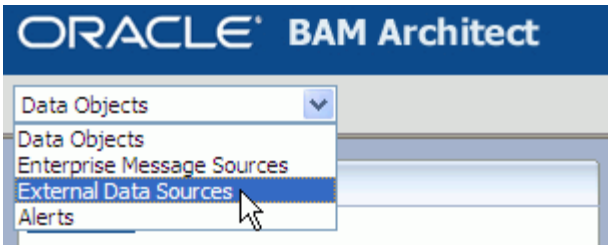
- If the ODI_Master or ODI_Work repositories are moved to different hosts after the initial installation, you must update the corresponding EDS definitions in Oracle BAM Architect.

54.6.1 How to Update the Oracle Data Integrator External Data Source Definitions

To update the Oracle Data Integrator external data source definitions:

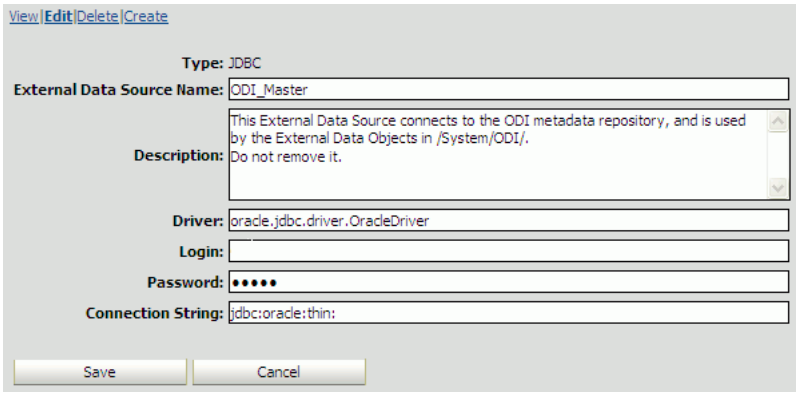
1. Open Oracle BAM Architect, and go to the External Data Sources page.

Figure 54–1 Opening External Data Source Page in Oracle BAM Architect



2. Select **ODI_Master** or **ODI_Work**, and click **Edit**.

Figure 54–2 Editing the ODI_Master External Data Source



3. Update the **Login**, **Password**, or **Connection String** parameters as needed, and click **Save**.

54.7 Launching Oracle Data Integrator Scenarios From Oracle BAM Alerts

Alerts created in Oracle BAM can launch Oracle Data Integrator scenarios when specified conditions are met. See Section F.3.10, "Run an Oracle Data Integrator Scenario" for more information.

54.8 Running Oracle Data Integrator Agent as a Daemon or a Microsoft Windows Service With Oracle BAM Embedded

There are several ways to run Oracle Data Integrator scenarios in which Oracle BAM functionality has been embedded. This section provides information about configuring Oracle BAM if you run the Oracle Data Integrator agent as a daemon or a Microsoft Windows Service.

1. On the Oracle BAM host, go to the `ORACLE_HOME\bam\ODI\tools\wrapper\conf` directory.
2. Copy the two files contained in that directory (`snpsagent.conf.bam` and `readme.txt`) to the host on which the Oracle Data Integrator agent runs as a daemon or service, in the `ODI_HOME\tools\wrapper\conf` directory.
3. Follow the instructions in the `readme.txt` file in that directory to configure the Oracle Data Integrator agent to run with Oracle BAM.

The `agent.bat` (or `agent.sh`) file picks up the same environment variables as do the other Oracle Data Integrator applications (such as Designer, Topology, Operator). As long as the Oracle Data Integrator integration installation has been performed on the Oracle Data Integrator directory in which the `agent` script runs, no additional steps are needed to run the Oracle Data Integrator agent as a standalone application or as a daemon or service.

55

Creating External Data Sources

This chapter contains the information needed to create and manage External Data Sources (EDS).

This chapter contains the following topics:

55.1 Introduction to External Data Sources

An External Data Source (EDS) is a connection to an external database. An EDS usually contains data that does not change very much or data that is too large to bring into the Oracle BAM Active Data Cache (ADC).

The EDS definition in Oracle BAM acts as a pointer to the external data. For example, looking up the customer name based on a customer code in a customer management system. The customer name-code mapping is fairly static so that bringing that external data into Oracle BAM is not required.

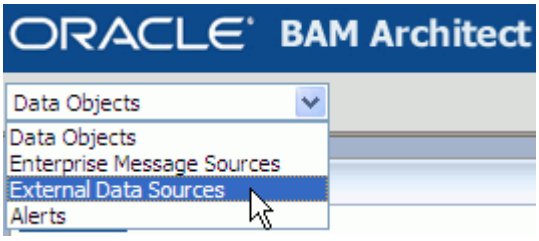
EDS definitions can be exported and imported using ICommand, but you cannot import or edit the contents using ICommand or Oracle BAM Architect.

Passwords are entered in clear text. You cannot use DSNs (data source names).

To view the existing EDS:

- Select **External Data Sources** from the Oracle BAM Architect function list.

Figure 55–1 Oracle BAM Architect Function List



55.2 Creating External Data Sources

Oracle BAM external data sources are created, edited, and deleted using Oracle BAM Architect.

55.2.1 How to Create an External Data Source

To define an EDS:

1. Select **External Data Sources** from the Oracle BAM Architect function list.
2. Click **Create**.
3. Enter a name and a description for the EDS.

   Caution: A single or double quotation mark in an Oracle BAM object name, such as a data object, report, or enterprise message source name, causes a runtime error.

   Do not include single or double quotation marks in an Oracle BAM object name.

4. Enter **Driver**, for example, `oracle.jdbc.driver.OracleDriver` for Oracle.
5. Enter database user credentials in the **Login** and **Password** fields.
6. Enter **Connection string/URL**, for example

   `jdbc:oracle:thin:@`*`db_host_name`*`:`*`db_port`*`:`*`db_instance`*

55.2.2 What You May Need to Know About Oracle Data Integrator External Data Sources

If you install the integration files for Oracle BAM and Oracle Data Integrator, two EDS definitions are created in Oracle BAM Architect: ODI_Master and ODI_Work. These EDS definitions cannot be deleted from Oracle BAM Architect, and their configuration should not be changed unless you are updating your Oracle Data Integrator host.

55.2.3 How to Edit an External Data Source

To edit an EDS:

1. Select **External Data Sources** from the Oracle BAM Architect function list.
2. Select the EDS to edit.

   The EDS properties display.
3. Select **Edit**.
4. Make the changes and click **Save**.

55.2.4 How to Delete an External Data Source

> **Note:** If the EDS definitions ODI_Master and ODI_Work appear in Oracle BAM Architect, do not delete them. These EDS definitions are used by the integration between Oracle BAM and Oracle Data Integrator

To delete an EDS:

1. Select **External Data Sources** from the Oracle BAM Architect function list.
2. Select the EDS to delete.

   The data source properties display.
3. Select **Delete**.
4. Click **OK** to confirm deletion of the data source.

   The data source is deleted.

55.3 External Data Source Example

This example uses the sample SCOTT user account and the EMP table in the Oracle database. You may need to unlock the account before proceeding with this example.

Step 1: Create an EDS

1. Select **External Data Sources** from the Oracle BAM Architect function list.
2. Click **Create**.
3. Enter `myDataSource` in the **External Data Source Name** field.
4. Enter `My Example External Data Source` in the **Description** field.
5. Enter `Microsoft ODBC for Oracle` in the **Driver** field.
6. Enter `scott` in the **Login** field and tiger in the **Password** field.

   This sample account comes with your Oracle database installation. If you do not have this sample account you can create a new account and use it for this example.
7. Enter `server=`*`net_service_name`* in the **Connection string/URL**.

   This entry must be a Net Service Name defined in your tnsnames.ora file.
8. Click **Save**.
9. Click **Continue**.

   The EDS information is displayed on the screen.

Step 2: Create a Data Object using the EDS

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Click **Create Data Object**.
3. Enter `Employees` in the **Name for new Data Object** field.
4. Leave the slash (/) in the **Location for new Data Object** field.

   The data object appears in the top level **Data Objects** folder.
5. Leave the **Tip text** field blank.

6. Enter `Oracle Database Sample EMP Table` in the **Description** field.
7. Select the **External Data Source** checkbox.
8. Select **myDataSource** from the **External Data Source** list.
9. Enter `emp` in the **External Table Name** field.
10. Add the following fields to the data object:

Table 55–1 Fields in Employees Data Object

Field	External Field Name	Field Type
ename	ename	String
empno	empno	Integer
job	job	String
mgr	mgr	Integer
hiredate	hiredate	DateTime
sal	sal	Decimal
comm	comm	Decimal
deptno	deptno	Integer

Keep default settings for field attributes not specified in the table.

11. Click **Create Data Object**.
12. Click **Continue**.
13. Click **Contents** to view the contents of the data object

The data in the Employees data object should match the data in the Oracle database sample EMP table.

55.4 Use Case: Creating an EDS Against Oracle Business Intelligence Enterprise Edition

The following are the steps to configure an EDS definition in Oracle BAM Architect to work with Oracle Business Intelligence Enterprise Edition.

1. Get the `bijdbc.jar` file and add it to the Oracle WebLogic Server class path.

   Add the JAR to `WEBLOGIC_CLASSPATH` in

   WLS_HOME`/wlserver 10.3/common/bin/commEnv.cmd`

2. Create an EDS in Oracle BAM Architect with the following details:

   Driver: `oracle.bi.jdbc.AnaJdbcDriver`

   Login: User name for the Oracle Business Intelligence Server

   Password: Password for the Oracle Business Intelligence Server

   Connection String/URL:
   `jdbc:oraclebi://`*host_name*`:`*port_number*`/catalog=`*catalog_name*`;`

   For example: `jdbc:oraclebi://bihost:9703/catalog=Paint;`

   See "Step 1: Create an EDS" on page 55-3 for an example EDS configuration.

3. Create a data object based on this EDS. See "Step 2: Create a Data Object using the EDS" on page 55-3 for an example.

56

Using Oracle BAM Web Services

The Oracle BAM web services are part of the Oracle BAM technologies that feeds data to the Oracle BAM Server. This chapter provides information about using the Oracle BAM web services.

This chapter includes the following sections:

56.1 Introduction to Oracle BAM Web Services

The Oracle BAM web services allow users to build applications that publish data to the Oracle BAM Server for use in real-time charts and dashboards. Any client that can talk to standard web services can use these APIs to publish data to Oracle BAM. The Oracle BAM web services interfaces allow integration of Oracle BAM with other components such as Oracle BPEL Process Manager and Oracle Mediator, and they facilitate SOA composite application development.

> **Note:** This option cannot be used for complex processing of messages, performing lookups in Oracle BAM Active Data Cache to augment the data, or initial bulk uploads to set up a star schema.

The data objects in the Oracle BAM Server are available using the Oracle BAM web services. There are several other meta objects that are available using the ICommand web service.

External web services can be called by an Oracle BAM alert rule. See Section 57.2, "Creating Alert Rules" for more information.

Oracle BAM provides the following static untyped web service APIs:

- **DataObjectOperations10131** allows clients developed for Oracle BAM 10.1.3.x servers to make web service calls to DataObjectOperations on Oracle BAM 11g servers.
- **DataObjectOperationsByID** allows developers to interact with data objects by their ID (for example, _Call_Center).

- **DataObjectOperationsByName** allows developers to interact with data objects by their display names (for example, Call Center).
- **DataObjectDefinition** performs operations to get, create, delete, and update definitions of Data Objects.
- **ManualRuleFire** is used by other Oracle BAM services to launch rules created in Oracle BAM Active Studio.
- **ICommand** is a DOS command-line utility that provides a set of commands that perform various operations on items in the Oracle BAM Server. The ICommand web service exposes all of the ICommand functionality through a web service.

These services can be discovered within an Oracle BAM Server using a WSIL interface.

56.2 Using the DataObjectOperations Web Services

The DataObjectOperations web service allows users to manipulate the Data Objects in the Oracle BAM Server by inserting, updating, deleting and upserting rows into the Data Objects.

The following operations are supported by the DataObjectOperations web service interfaces.

- **Batch** performs batch operations on a data object. Batch is not supported for DataObjectOperationsByName web service.
- **Delete** removes a row from the data object.
- **Get** fetches the details from a data object per the specifications in the XML payload. Get is only available in DataObjectOperationsByName web service.
- **Insert** adds a row to the data object.
- **Upsert** inserts new data into an existing row in a data object if the row exists. If the row does not exist a new row is created.
- **Update** inserts new data into an existing row in a data object.

The request and response messages vary depending on the operation used. See Section E.1, "DataObjectOperations10131," Section E.2, "DataObjectOperationsByName," and Section E.3, "DataObjectOperationsByID" for information about using the operations supported by each of the web services.

56.2.1 How to Use the DataObjectOperations Web Services

To use the DataObjectOperations web service, create a web service proxy in your application in Oracle JDeveloper.

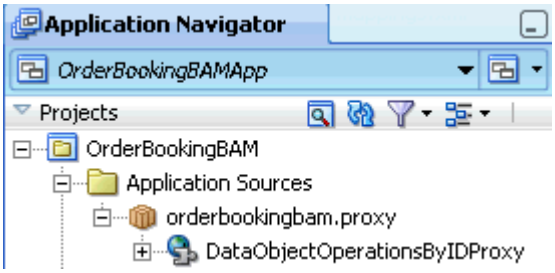
The Web Services Description Language (WSDL) files for the DataObjectOperations web services are available at the following URLs on the system where Oracle BAM web services are installed.

```
http://host_
name:7001/OracleBAMWS/Services/DataObject/DataObjectOperations.asmx?WSDL

http://host_name:7001/OracleBAMWS/WebServices/DataObjectOperationsByID?WSDL

http://host_name:7001/OracleBAMWS/WebServices/DataObjectOperationsByName?WSDL
```

Note: The default port for Oracle BAM web services on the Administration Server is 7001. On managed servers the default port number is 9001.

When the web service proxy is created, you see it in the Application Navigator under the Application Sources folder in your project as shown in Figure 56–1.

Figure 56–1 DataObjectOperations Web service proxy in Application Sources



56.3 Using the DataObjectDefinition Web Service

The DataObjectDefinition web service allows a web service client to create, update, delete, and get data object definitions.

The following operations are supported by DataObjectDefinition web service.

- **Create** creates a data object. For more information see Section E.4.1, "Create."
- **Delete** removes a data object from the server. For more information see Section E.4.2, "Delete."
- **Get** returns the definition of an existing data object. For more information see Section E.4.3, "Get."
- **Update** changes the definition of a data object. For more information see Section E.4.4, "Update."

The request and response messages vary depending on the operation used. See Section E.4, "DataObjectDefinition Operations" for more information.

56.3.1 How to Use the DataObjectDefinition Web Service

To use the DataObjectDefinition web service you create a web service proxy in your application in Oracle JDeveloper.

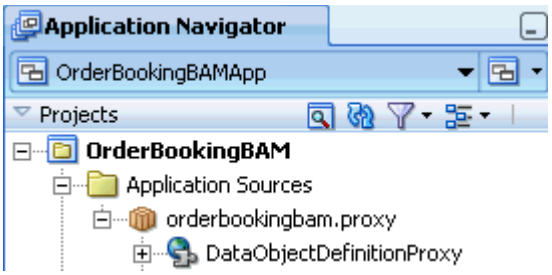
The WSDL file for the DataObjectDefinition web service is available at the following URL on the system where Oracle BAM web services are installed.

`http://host_name:7001/OracleBAMWS/WebServices/DataObjectDefinition?WSDL`

Note: The default port for Oracle BAM web services on the Administration Server is 7001. On managed servers the default port number is 9001.

When the web service proxy is created, you see it in the Application Navigator under the Application Sources folder in your project as shown in Figure 56–2.

Figure 56–2 DataObjectDefinition Web service proxy in Application Sources



56.4 Using the ManualRuleFire Web Service

The ManualRuleFire web service allows users to launch rules in the Oracle BAM Server. FireRuleByName is the available operation. See Section E.5, "ManualRuleFire Operations" for details.

56.4.1 How to Use the ManualRuleFire Web Service

To use the ManualRuleFire web service, you create a web service proxy in your application in Oracle JDeveloper.

The WSDL file for the ManualRuleFire web service is available at the following URL on the system where Oracle BAM web services are installed.

`http://host_name:7001/OracleBAMWS/WebServices/ManualRuleFire?WSDL`

Note: The default port for Oracle BAM web services on the Administration Server is 7001. On managed servers the default port number is 9001.

When the web service proxy is created, you see it in the Application Navigator under the Application Sources folder in your project.

56.5 Using the ICommand Web Service

ICommand is available as a web service for application developers who want to interact with ICommand features over HTTP.

The ICommand web service includes most of the same features as the command-line utility. For example, you can use it to:

- Delete a data object
- Import rows into a data object
- Export a report

The key differences revolve around the fact that the web service cannot access files on the remote system. Therefore, you cannot pass in a file name when using the `import` command or the `export` command.

Instead, you must pass in the `import` content inline. Similarly, you receive the `export` content inline.

Commands other than `import` and `export` generally work the same as with the command-line utility.

For more information about the commands and parameters provided by ICommand, see Appendix G, "Oracle BAM ICommand Operations and File Formats."

The ICommand web service has a single method, called `Batch`. It takes a single input parameter, which is a string containing a set of commands in the syntax described in Section G.3, "Format of Command File." The return value is a string containing the results of executing each command, in the log syntax described in Section G.4, "Format of Log File."

56.5.1 How to Use the ICommand Web Service

The WSDL file for the ICommand web service is available on the system where Report Server has been installed. It is available at the following URL:

`http://host_name:7001/OracleBAMWS/WebServices/ICommand?WSDL`

Note: The default port for Oracle BAM web services on the Administration Server is 7001. On managed servers the default port number is 9001.

Example 56–1 Deleting a Data Object (Input)

```
<OracleBAMCommands>
  <Delete type="dataobject" name="/test123"/>
</OracleBAMCommands>
```

57

Creating Oracle BAM Alerts

This chapter describes how to create alerts in Oracle BAM.

This chapter contains the following topics:

57.1 Introduction to Creating Alerts

Alerts are launched by a set of specified events and conditions, known as a *rule*. Alerts can be launched by data changing in a report or can send a report to users daily, hourly, or at set intervals. *Events* in an alert rule can be an amount of time, a specific time, or a change in a specific report. *Conditions* restrict the alert rule to an event occurring between two specific times or dates. As a result of events and conditions, reports can be sent to users through email.

Alerts can be created in both the Oracle BAM Architect and Oracle BAM Active Studio web applications.

Alerts are shown in the Alert Rules table. In Oracle BAM Active Studio the table includes a Last Launched column that indicates the last time the alert rule was fired. Each alert name is accompanied by an icon indicating its status as described in Table 57–1.

Figure 57–1 Alert Rules Table in Oracle BAM Architect

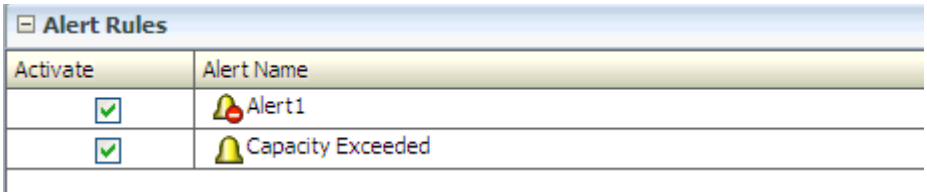


Table 57–1 Alert Rule Icons

Icon	Description
	**Normal** indicates that the alert is active and fires under the conditions specified in the rule.
	**Invalid** indicates that an alert has become orphaned or broken due to some error. This icon is displayed when an alert cannot be loaded properly into the Event Engine. The rule might require correction. For example, when a report is deleted and an alert based on this report still exists, that alert cannot be loaded properly. This icon appears only when rules are loaded into the Event Engine (on restarts). Alerts displayed with this icon do not fire again until they are edited and corrected.
	**Expired** means that the alert does not fire again. This icon is seen in time based alerts which fire only one time, after the alert has fired. However, these alerts can be edited and reused, resetting the state to Normal.

Note that inactive and expired alerts behave differently. An alert can be deactivated only if it is running. This behavior is a benefit to users who do not want to receive alerts for some time interval, but want to retain the ability to activate the alert at a convenient time. Alerts that are not active, but still valid (displayed with the Normal icon) can be activated again.

Those alerts that are expired have run for the specified condition and do not run again. They cannot be activated to run again. However, if you want to reuse an expired alert, double click the alert, update the definition to make it a valid rule, and save the alert rule definition. The alert is reloaded and is ready to fire again.

Note: If any changes to the time or time zone are made on the Oracle BAM Server system, the Oracle BAM Server application must be restarted or time-based alerts misfire.

57.2 Creating Alert Rules

A rule specifies the events and conditions under which an alert fires.

Note: An alert fires only if its triggering event conditions are met from the point in time the alert is defined (or reenabled) and forward. An alert does not fire if its conditions were met before it was defined, or while it was disabled.

57.2.1 How to Create an Alert Rule

This section describes how to create Oracle BAM alert rules in Oracle BAM Architect. The procedure is the same in Oracle BAM Active Studio.

To create a rule:

1. Select **Alerts** in the Oracle BAM Architect function list.

   In Oracle BAM Active Studio, select the **Alerts** tab.

2. Click **Create A New Alert**.

The Rule Creation and Edit dialog box opens.

3. Click **Create A Rule.**
4. Enter a name for the rule.

> **Caution:** A single or double quotation mark in an Oracle BAM object name, such as a data object, report, or enterprise message source name, causes a runtime error.
>
> Do not include single or double quotation marks in an Oracle BAM object name.

5. Select an event that launches the alert.

   See Section F.1, "Events" for descriptions of each event.
6. Click **Next**.
7. Select one or more conditions, if needed.

   See Section F.2, "Conditions" for descriptions of each condition.
8. Select one or more actions. See Section F.3, "Actions" for descriptions of each action.
9. In the rule expression, click each underlined item and specify a value to complete the alert rule.

   For example, click **select report**, and choose a report in the dialog box that opens. Other values you define include user names receiving reports, dates and times, time intervals, and filter expressions for a specific field. To continue adding conditions or actions, click the last line in the expression and then select another condition or action.

   You can click the **Back** and **Next** buttons to go between the events page and the page containing actions and conditions, and make changes to those parts of the rule expression you have constructed.
10. You can click the **Frequency Constraint** button to set a limit to how often an alert can launch.

    The default frequency constraint for alerts is five seconds. Type a number and select a time measurement such as seconds, minutes, or hours, and click **OK**. To turn off the frequency constraint, uncheck the **Constraint Enabled** checkbox. For more information about frequency constraint see Section F.4, "Frequency Constraint."
11. Click **Delete this expression** to remove lines from the alert rule.
12. Click **OK**.

    The alert rule is added to list and is active.

57.2.2 How to Activate Alerts

When you create an alert rule, it is automatically active. If you want an alert to be temporarily inactive but you do not want to delete it, you can turn it off by deselecting the **Activate** checkbox.

To change the activity status of an alert rule:

1. Select **Alerts** from the Oracle BAM Architect function list.

2. Select the **Activate** checkbox for the alert rule.

A checked box means the alert rule is active.

An unchecked box means the alert rule is inactive.

Selecting the **Activate** checkbox does not cause an alert to launch, it only enables the rule so that if the specified event occurs, the alert launches.

An exclamation mark on the alert icon indicates it has launched and is not valid again, or because items that it references are missing and it cannot launch.

57.2.3 How to Modify Alert Rules

When you modify alert rules created from a template, you can add new lines and select conditions and actions the same as when you build alert rules without templates.

To modify an alert rule:

1. Select the alert rule to edit.
2. Click **Edit** in the Alert Actions list.

   The Rule Creation and Edit dialog box opens.
3. Make changes to the alert and click **OK**.

57.2.4 How to Delete an Alert

To delete an alert:

1. Select the alert to delete.
2. Click **Delete** in the Alert Actions list.

   A dialog box opens to confirm alert deletion.
3. Click **OK**.

   The alert is deleted.

57.3 Creating Alert Rules From Templates

Alert rule templates are a convenient preselected group of events and conditions based on some common use cases.

57.3.1 How to Create Alert Rules From Templates

To create an alert rule from a template:

1. Click **Create A New Alert**.

   The Create Alert Rule dialog box opens.
2. Click **Create A Rule From A Template**.
3. Enter a name for the alert rule.
4. Select a template from the list.
5. In the **Rule Expression** box, click each underlined item and specify a value to complete the alert rule. For example, click **select report**, and choose a report in the

dialog box that opens. Other values you define include user names receiving reports, dates and times, time intervals, and filter expressions for a specific field.

6. You can click **Frequency Constraint** to specify how often an alert can launch. The default frequency constraint for alerts is five seconds. Enter a number and select a time measurement such as seconds, minutes, or hours, and click **OK**.
7. You can click **Modify this rule** to modify the rule without using the template. This provides more options for creating rules.
8. Click **OK**.

   The alert rule is added to list and is active.

57.4 Creating Alert Rules With Messages

You can create alert rules that send messages. The messages can contain information such as report names, links to reports, and user names. Messages can also include variables that are set when the alert is launched, such as the time that an event occurred and the data that launched the event. To use data variables, the event must be based on data.

57.4.1 How to Create an Alert Rule With a Message

You can create alert rules that send messages. The messages can contain information such as report names, links to reports, and user names. Messages can also include variables that are set when the alert is launched, such as the time that an event occurred and the data that launched the event. To use data variables, the event must be based on data.

To create an alert rule that includes a message:

1. Start building an alert rule.
2. Select the action **Send a message via email**.
3. Click **create message** in the rule expression.

   The Alert Message dialog box opens.
4. Enter a subject in the **Subject** line.
5. Enter the message in the **Message Text** box.
6. Include special fields into the message.

   Special fields are listed in the box in the lower left corner of the Alert Message dialog box. The special fields listed change when reports are selected on the right side of the dialog box.

   To insert a special field into the message:

   a. Select a special field from the list.

   b. Click **Insert into subject** or **Insert into text**.

   You can insert multiple values of the same type, for example, multiple links to different reports.

   - **Send Report Name** inserts name of selected report.
   - **Send Report Owner** inserts owner name of selected report.
   - **Send Report Link** inserts link to selected report.

- **Changed Report Name** inserts name of the changed report.
- **Changed Report Owner** inserts Owner Name Of Changed Report.
- **Target User** inserts user name of message recipient.
- **Date/Time Sent** inserts date and time of message sent.

7. Click **OK**.

57.5 Creating Complex Alerts

You can create nested rules with many actions and chained rules that launch other rules.

You can chain rules by creating two types of rules:

- A dependent rule that must be launched by another rule.
- A rule with an action to launch a dependent rule.

57.5.1 How to Create a Dependent Rule

To create dependent rules:

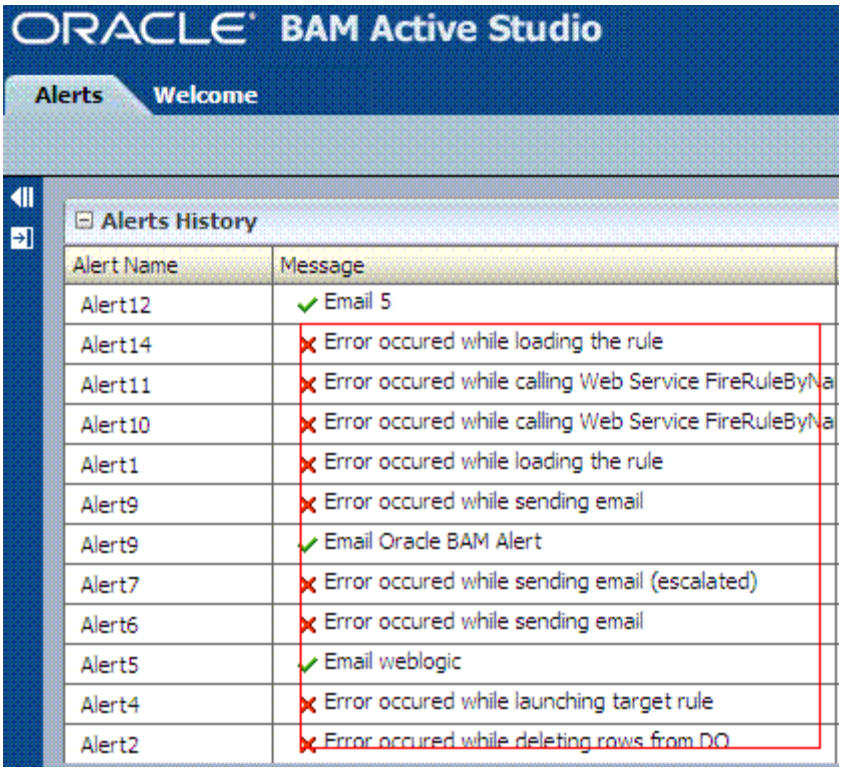
1. Create a rule that includes the event **When this rule is launched**. No value is required for this event.
2. Create a rule that includes the action **Launch a rule** or **Launch rule if an action fails**. The **Launch rule if action fails** applies to any of the actions contained in the rule.
3. Click **select rule** in the action.

   The Select Dependent Rule dialog box opens.
4. Select a dependent rule. Only rules that include the **When this rule is launched** event are displayed in the list.
5. Click **OK**.

To handle a failing action, add the action **Launch rule if action fails**. For example, if a rule is supposed to send a message, and for some reason the message does not send, you could launch another rule to notify you.

57.6 Using Alert History

Alert history is available in Oracle BAM Active Studio providing a list of alert rules triggered and their status messages.



57.6.1 How to View Alert History

You can view recent history of alert activity on the Alerts tab in Oracle BAM Active Studio. The Alerts History list displays the 25 most recent alerts launched.

In the Alerts History list, you can view the names of recently launched alerts, any messages associated with the alerts, the users who created the alerts, and the time and date that the alert rules were triggered.

In the case of alert rules that send e-mail, the Alerts History list only displays the alert if the user currently logged in is an alert e-mail recipient. It is not listed in the Alerts History list--even if the user is the creator of the alert--if the user is not a recipient of the alert.

However, if an alert fails to send a message to an alert recipient, the message is logged with the alert owner's name, so that the owner can see the error message in the Alerts History pane and take corrective action if necessary. A non-existing name cannot be logged as the alert recipient's name.

Alerts History Messages

The **Message** column of the Alerts History list provides information about the success or failure of alert delivery. The successful alert is shown with a green checkmark next to the message. The unsuccessful alert is displayed with a red **x** icon and a message indicating how the alert failed at the time of loading or processing. Click the **x** icon for additional information about the error.

If a report is deleted that is referenced by an alert, there is no warning to the user. When the alert is triggered, the error message "Error occurred while sending e-mail" is given with no specific error regarding broken references to the deleted report. When deleting reports, it is important to verify that the report is not referenced by an alert, or this error occurs.

57.6.2 How to Clear Alert History

When many alerts are actively launching and the alert history list becomes long, you might want to clear your alert history list.

To clear the alert history:

1. On the Alerts tab, click **Clear alert history**.

   A message is displayed to confirm to clear alert history.

2. Click **OK**.

   The alert history list is deleted. New alerts launched after clearing appear in the alert history list.

57.7 Launching Alerts by Invoking Web Services

You can use the alerts web service to manually launch alerts. For more information, refer to:

`http://host:http_port/OracleBAMWS/WebServices/ManualRuleFire?wsdl`

You define the rule name using the format:

`username.alertname`

Note: Oracle BAM Active Studio URLs used in alerts and report links contain a virtual directory using the product build number for caching and performance purposes. This directory must be included in links, and it is not recommended to edit these links. Links created with a previous version of Oracle BAM do not work after a product upgrade. The alert requires editing or the report shortcut must be copied again.

57.8 Calling an External Action

Call an External Action is used to develop a custom action. For users whose requirements cannot be fulfilled by the actions provided by Oracle BAM, this feature is used to extend the action set.

External actional actions are not seen in the Oracle BAM Alerts user interface by default. They must be registered with Oracle BAM before they are seen in the user interface.

To do this task, the `EventEngine` interface must be implemented and you must develop an action around it. That means you must write Java code, bundle the compiled code in a JAR file. Then register it in Oracle BAM Architect as an action in the System/Alerts/External Actions data object.

Call an External Action action is not required to invoke Web services. The action was used in this way in pre 11g releases, but was replaced by Call a Web Service action in Oracle BAM 11g. Call a Web Service action has a more sophisticated Web services client, which is dynamic and can invoke any service by reading WSDLs at runtime.

57.9 Sending Alerts to External E-mail Accounts

Alerts from Oracle BAM can be sent to e-mail accounts that are unknown to Oracle BAM if a property is set in the Oracle BAM common configuration file.

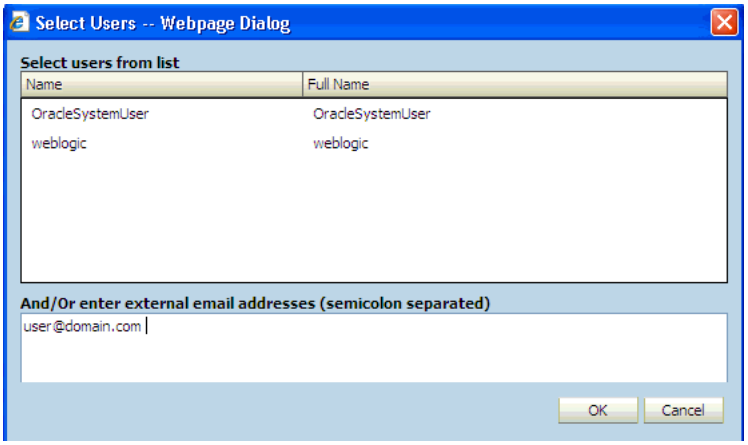
This feature is available only for the actions **Send a report via email** and **Send a message via email**.

To send alerts to external e-mail accounts:

1. Set the property `AlertActionAllowExternalEmail` to true in the `BAMCommonConfig.xml` configuration file.

   See "Configuring Advanced Properties" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for information about editing Oracle BAM configuration files.

2. Restart Oracle BAM.
3. Create an alert rule containing an action that sends messages to users.
4. Click **select user** and enter the e-mail addresses in the **And/Or external email addresses** box, separating the addresses with semicolons.



58 Using ICommand

This chapter provides usage information for the ICommand command-line utility.

This chapter includes the following sections:

- Section 58.1, "Introduction to ICommand"
- Section 58.2, "Executing ICommand"
- Section 58.3, "Specifying the Command and Option Syntax"
- Section 58.4, "Using Command-line-only Parameters"
- Section 58.5, "Running ICommand Remotely"

58.1 Introduction to ICommand

ICommand is a command-line utility (and web service) that provides a set of commands that perform various operations on items in the Active Data Cache. You can use ICommand to export, import, rename, clear, and delete items from Active Data Cache. The commands can be contained in an input XML file, or a single command can be entered on the command line. Informational and error messages may be output to either the command window or to an XML file.

For more information about using the ICommand web service, see Section 56.5, "Using the ICommand Web Service."

For information about individual commands and their parameters see Appendix G, "Oracle BAM ICommand Operations and File Formats."

58.2 Executing ICommand

ICommand can be executed using the *ORACLE_HOME*\bam\bin\icommand.bat file on the Microsoft Windows platform and *ORACLE_HOME*\bam\bin\icommand.sh shell script on UNIX platforms.

Just entering `icommand` on the command line provides the user with a summary of the ICommand operations and parameters.

Before attempting to execute ICommand, the `JAVA_HOME` environment variable must be set to point to the root directory of the supported version of Java Development Kit (see the Oracle BAM support matrix on Oracle Technology Network web site for supported JDK versions).

Note: When Oracle BAM is installed, ICommand looks for the Oracle BAM Server on port 9001 by default. If the Oracle BAM Server port number is changed from the default during the setup and configuration of Oracle BAM, then the user must manually change the port number from 9001 to the new port number in the file `BAMICommandConfig.xml`.

The property to change is

```
<ServerPort>9001</ServerPort>
```

The `BAMICommandConfig.xml` file is located in *`WLS_HOME`*`/user_projects/domains/base_domain/config/fmwconfig/servers/bam_server1/applications/oracle-bam_11.1.1/config/`.

58.3 Specifying the Command and Option Syntax

The basic structure of the ICommand command line entry is as follows:

```
icommand -username user_name -cmd command_name -name value -type value [-parameter value]
```

All parameters given on the command line are in the following form:

```
-parameter value
```

The `parameter` portion is not case sensitive. If the `value` portion contains spaces or other special characters, it must be enclosed in double quotation marks. For example

```
icommand -cmd export -name "/Samples/Call Center" -type dataobject
 -file C:\CallCenter.xml
```

It is required to use quotation marks around report names and file names that contain spaces and other special characters.

For some parameters, the `value` may be omitted. See Section G.2, "Detailed Operation Descriptions," for information about individual parameter values.

58.3.1 How to Specify the Security Credentials

ICommand requires users to provide security credentials when running operations. If no security credentials have been specified in the configuration file, ICommand securely prompts for a user name and password.

To use default credentials, add the `ICommand_Default_User_Name` and `ICommand_Default_Password` properties to the *`WLS_HOME`*`/user_projects/domains/base_domain/config/fmwconfig/servers/bam_server1/applications/oracle-bam_11.1.1/config/BAMICommandConfig.xml` file. For example:

```
<ICommand_Default_User_Name>user_name</ICommand_Default_User_Name>
<ICommand_Default_Password>password</ICommand_Default_Password>
```

However, command line entries always override the properties specified in the configuration file.

The user name and password for running ICommand operations can come from the configuration file, command line prompts, or command line options as follows:

- If the user name and password are only specified in the configuration file (that is, `-username` parameter is not used in the command line), then the `ICommand_Default_User_Name` and `ICommand_Default_Password` values in the configuration file are used.
- If only the user name is specified in the configuration file and the password is not, then the user name value is used, and ICommand prompts the user for the password at the command line.
- If user name is specified on the command line, then that value is used, and ICommand prompts the user for a password. The password prompt occurs regardless of any properties specified in the configuration file. For example:

  ```
  icommand -cmd export -name TestDO -file C:\TestDO.xml -username user_name
  ```

58.3.2 How to Specify the Command

On the command line, commands are specified by the value of the `cmd` parameter. Options for the command are specified by additional parameters. For example

```
icommand -cmd export -name TestDO
 -type dataobject -file C:\TestDO.xml
```

In an XML command file, commands are specified by the XML tag. Options for the command are given as XML attribute values of the command tag, in the form `parametername=`*`value`*.

Command names and parameter values (except for Active Data Cache item names) are not case sensitive.

For information about individual commands and their parameters see Appendix G, "Oracle BAM ICommand Operations and File Formats."

58.3.3 How to Specify Object Names

Whenever an object name is specified in a command, the following rules apply.

General rules

When specified on a command line, if the name contains spaces or characters that have special meaning to DOS or UNIX, the name must be quoted according to the rules for command lines.

When specified in an XML command file, if the name contains characters that have special meaning within XML, the standard XML escaping must be used.

Data Objects

If the Data Object is not at the root, the full path name must be given, as in the following example:

```
/MyFolder/MySubfolder/MyDataObject
```

If the Data Object is at the root, the leading slash (/) is optional. The following two examples are equivalent:

```
/MyDataObject
MyDataObject
```

Data Object Folders

To specify a folder in Data Objects you must include the prefix `/public/DataObject/` at the beginning of the path to the folder.

```
/public/DataObject/MyFolder/MySubfolder
```

Reports and Report Folders

The full path name plus the appropriate prefix must be specified as in the following examples.

For shared reports the `/public/Report/` prefix must be included as shown here:

```
"/public/Report/Subfolder1/My Report"
```

For private reports the `/private:`*`user_name`*`/Report/` prefix must be included:

```
"/private:jsmith/Report/Subfolder1/My Report"
```

The `/private:`*`user_name`*`/` part of the prefix may be omitted if the user running ICommand is the user that owns the report.

```
"Report/Subfolder1/My Report"
```

The path information without the `public` or `private` prefix is saved in the export file.

Similarly, a report folder can be specified using the appropriate prefix.

```
/public/Report/Subfolder1

/private:jsmith/Report/Subfolder1
```

Alert Rules

Either the name of the Alert, or the full name of the Alert may be specified. The following two examples are equivalent for Alerts if the user running ICommand is the user that owns Alert1:

```
Alert1

/private:user_name/Rule/Alert1
```

If the user running ICommand is not the owner of Alert1, then only the second form may be used.

All other object types

Specify the full name of the object.

58.3.4 How to Specify Multiple Parameter Targets

Instead of creating a separate command line for each Active Data Cache object type, such as Dataobject, Folder, Report, and Rule, on which to execute a particular command, ICommand enables you to pass parameter values to several object types in the same command line.

For example:

```
icommand -cmd export -type all -report,rule,folder:owner 1
-dataobject,folder:permissions 1 -systemobjects 1 -file filename.xml
```

In this example, while exporting all of the objects in the system, the command passes owner = 1 to the report, rule, and folder Active Data Cache object types. The command also passes permissions = 1 to the dataobject and folder object types. The comma (,) separates the object types and the parameter is listed after a colon (:).

Supplying multiple values in the example single command line gives the same results as the following three commands:

```
icommand -cmd export -type report -owner 1 ...
icommand -cmd export -type rule -owner 1 ...
icommand -cmd export -type folder -owner 1 ...
```

58.4 Using Command-line-only Parameters

The following parameters can appear only on the command line:

- `Cmd`

  `-cmd` *`commandname`*

  Optional parameter that specifies a single command to be executed. Any parameters needed for the command must also be on the command line.

  The `Cmdfile` and `cmd` parameters are mutually exclusive. Exactly one of them must be present.

- `Cmdfile`

  `-cmdfile` *`file_name`*

  Optional parameter that specifies the name of the file that contains commands to be processed. Because this is an XML file, it would usually have the XML extension, although that is not required.

  The `Cmdfile` and `cmd` parameters are mutually exclusive. Exactly one of them must be present.

- `Debug`

  `-debug` *`flag`*

  Optional parameter that indicates whether extra debugging information is to be output if there is an error. Any value other than `0` (zero), or the absence of any value, indicates that debugging information is to be output. If this parameter is not present, no debugging information is output.

- `Domain`

  `-domain` *`domain_name`*

  Optional parameter that specifies the domain name to use to login to the Active Data Cache (the name of the computer on which the Active Data Cache server is running).

  If this parameter is omitted, `main` is used, which means the server information is obtained from the `ServerName` property in the `ICommand.exe.config` file.

  If the reserved value `ADCInProcServer` is used, then ICommand directly accesses the Active Data Cache database (which must be local on the same system on which ICommand is running) rather than contacting the Active Data Cache server. This option is necessary **only** when the Active Data Cache server is not running; otherwise corruption of the database could occur. The information about

the location and structure of the Active Data Cache database is obtained from various keys in the ICommand.exe.config file.

- `Logfile`

  `-logfile` *`file_name`*

  Optional parameter that specifies the name of the file to which results and errors are logged. If the file does not exist, it is created. If the file does exist, any contents are overwritten. Because this is an XML file, it would usually have the XML extension, although that is not required.

  If this parameter is not present, results and errors are output to the console.

  See Section G.4, "Format of Log File" for more information about the log file format.

- `Logmode`

  `-logmode` *`mode`*

  Optional parameter that indicates whether an existing log file is to be overwritten or appended to. The possible values for this parameter are `append` or `overwrite`. In either case, if the log file does not exist it is created.

  If this parameter is not present, `overwrite` is assumed.

  Note that because it is XML that is being added to the log file, if the `append` option is used the XML produced may not be strictly legal, as there is no top level root tag in the XML produced by successive appends (ICommand appends the same tag each time it is run). It is left up to the user to handle this.

- `Username`

  `-username` *`user_name`*

  Optional parameter that specifies the username that the command should run as. There is no password parameter.

  ICommand requires users to specify security credentials when running commands. ICommand securely prompts for a user name and password. If the `-username` parameter is specified on the command line, ICommand prompts the user for the password only.

58.5 Running ICommand Remotely

You can run ICommand from a remote system (where Oracle BAM is installed) and execute the commands on a server located remotely. To run ICommand remotely, add the properties ServerName and ServerPort in *`WLS_HOME`*`/user_projects/domains/base_domain/config/fmwconfig/servers/bam_server1/applications/oracle-bam_11.1.1/config/BAMICommandConfig.xml`, as shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BAMICommand>
  <ServerName>host_name</ServerName>
  <ServerPort>7001</ServerPort>
  <Communication_Protocol>t3</Communication_Protocol>
  <SensorFactory>oracle.bam.common.statistics.noop.SensorFactoryImpl</SensorFactor
y>
  <GenericSatelliteChannelName>invm:topic/oracle.bam.messaging.systemobjectnotific
ation</GenericSatelliteChannelName>
```

```
</BAMICommand>
```

The Oracle BAM version installed on the remote system should be same as the Oracle BAM Server version (that is, both servers should be from the same label).

Part XI

Using Oracle User Messaging Service

This part describes how to use Oracle User Messaging Service.

This part contains the following chapters:

59

Oracle User Messaging Service

This chapter describes Oracle User Messaging Service (UMS).

This chapter includes the following section:

59.1 Introduction to User Messaging Service

Oracle User Messaging Service enables two-way communication between users and deployed applications. Key features include:

- Support for a variety of messaging channels—Messages can be sent and received through Email, IM (XMPP), SMS (SMPP), and Voice. Messages can also be delivered to a user's SOA/WebCenter Worklist.
- Two-way Messaging—In addition to sending messages from applications to users (referred to as *outbound* messaging), users can initiate messaging interactions (inbound messaging). For example, a user can send an email or text message to a specified address; the message is routed to the appropriate application which can then respond to the user or invoke another process according to its business logic.
- User Messaging Preferences—End users can use a web interface to define preferences for how and when they receive messaging notifications. Applications immediately become more flexible; rather than deciding whether to send to a user's email address or instant messaging client, the application can simply send the message to the user, and let UMS route the message according to the user's preferences.
- Robust Message Delivery—UMS keeps track of delivery status information provided by messaging gateways, and makes this information available to applications so that they can respond to a failed delivery. Or, applications can specify one or more *failover* addresses for a message in case delivery to the initial address fails. Using the failover capability of UMS frees application developers from having to implement complicated retry logic.
- Pervasive integration within Fusion Middleware: UMS is integrated with other Fusion Middleware components providing a single consolidated bi-directional user messaging service.
  - Integration with Oracle BPEL—Oracle JDeveloper includes pre-built BPEL activities that enable messaging operations. Developers can add messaging capability to a SOA composite application by dragging and dropping the necessary activity into any workflow.

- Integration with Oracle Human Workflow—UMS enables the Human Workflow engine to send actionable messages to and receive replies from users over email.
- Integration with Oracle BAM—Oracle BAM uses UMS to send email alerts in response to monitoring events.
- Integration with Oracle WebCenter—UMS APIs are available to developers building applications for Oracle WebCenter Spaces. The API is a realization of Parlay X Web Services for Multimedia Messaging, version 2.1, a standard web service interface for rich messaging.

59.1.1 Components

There are three types of components that comprise the Oracle User Messaging Service. These components are standard Java EE applications, making it easy to deploy and manage them using the standard tools provided with Oracle WebLogic Server.

- UMS Server: The UMS Server orchestrates message flows between applications and users. The server routes outbound messages from a client application to the appropriate driver, and routes inbound messages to the correct client application. The server also maintains a repository of previously sent messages in a persistent store, and correlates delivery status information with previously sent messages.
- UMS Drivers: UMS Drivers connect UMS to the messaging gateways, adapting content to the various protocols supported by UMS. Drivers can be deployed or undeployed independently of one another depending on what messaging channels are available in a given installation.
- UMS Client applications: UMS client applications implement the business logic of sending and receiving messages. A UMS client application might be a SOA application that sends messages as one step of a BPEL workflow, or a WebCenter Spaces application that can send messages from a web interface.

In addition to the components that comprise UMS itself, the other key entities in a messaging environment are the external gateways required for each messaging channel. These gateways are not a part of UMS or Oracle WebLogic Server. Since UMS Drivers support widely-adopted messaging protocols, UMS can be integrated with existing infrastructures such as a corporate email servers or XMPP (Jabber) servers. Alternatively, UMS can connect to outside providers of SMS or text-to-speech services that support SMPP or VoiceXML, respectively.

59.1.2 Architecture

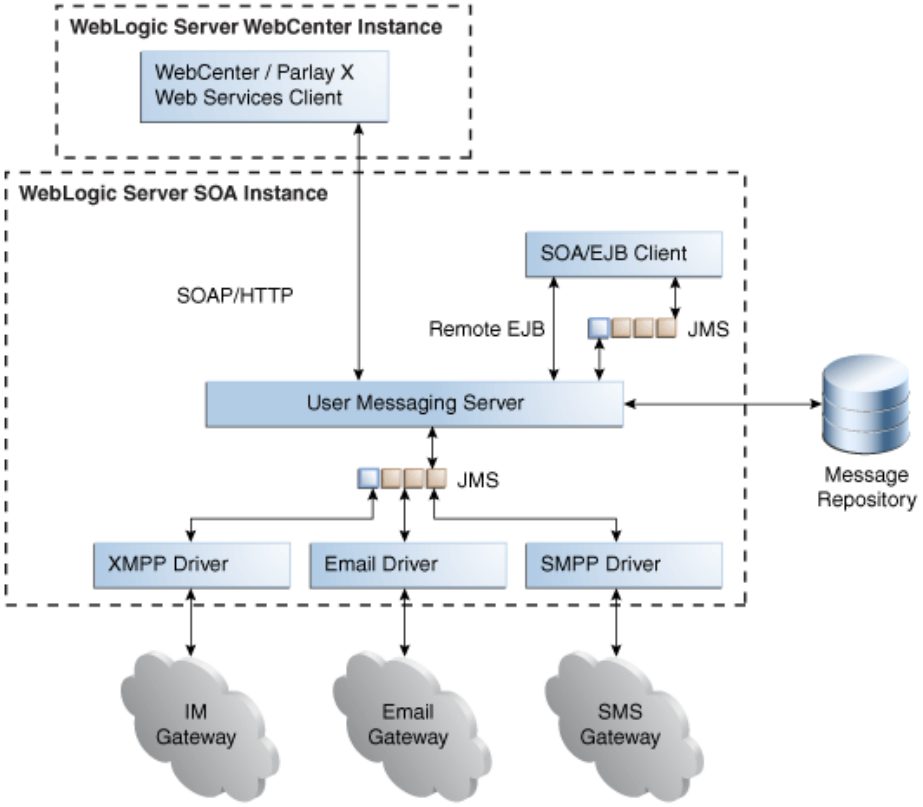
The system architecture of Oracle User Messaging Service is shown in Figure 59–1.

For maximum flexibility, the components of UMS are separate Java EE applications. This allows them to be deployed and managed independently of one another. For example, a particular driver can be stopped and reconfigured without affecting message delivery on all other channels.

Exchanges between UMS client applications and the UMS Server occur as SOAP/HTTP web service requests for web service clients, or through Remote EJB and JMS calls for BPEL messaging activities. Exchanges between the UMS Server and UMS Drivers occur through JMS queues.

Oracle UMS server and drivers are installed alongside SOA or BAM in their respective WebLogic Server instances. A WebCenter installation includes the necessary libraries to act as a UMS client application, invoking a server deployed in a SOA instance.

Figure 59–1 UMS architecture



60

Sending and Receiving Messages using the User Messaging Service EJB API

This chapter describes how to use the User Messaging Service (UMS) EJB API to develop applications, and describes how to build two sample applications, `usermessagingsample-ejb.ear` and `usermessagingsample-echo-ejb.ear`.

> **Note:** The User Messaging Service EJB API (described in this chapter) is deprecated. Use the User Messaging Service Java API instead, as described in Chapter 61, "Sending and Receiving Messages using the User Messaging Service Java API".

This chapter includes the following sections:

- Section 60.1, "Introduction to the UMS Java API"
- Section 60.2, "Creating a UMS Client Instance"
- Section 60.3, "Sending a Message"
- Section 60.4, "Receiving a Message"
- Section 60.5, "Using the UMS Enterprise JavaBeans Client API to Build a Client Application"
- Section 60.6, "Using the UMS Enterprise JavaBeans Client API to Build a Client Echo Application"
- Section 60.7, "Creating a New Application Server Connection"

> **Note:** To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL:
> `https://soasamples.samplecode.oracle.com/`
>
> Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

60.1 Introduction to the UMS Java API

The UMS Java API supports developing applications for Enterprise JavaBeans clients. It consists of packages grouped as follows:

- Common and Client Packages

  - `oracle.sdp.messaging`
  - `oracle.sdp.messaging.filter`: A MessageFilter is used by an application to exercise greater control over what messages are delivered to it.
- User Preferences Packages
  - `oracle.sdp.messaging.userprefs`
  - `oracle.sdp.messaging.userprefs.tools`

60.1.1 Creating a Java EE Application Module

There are two choices for a Java EE application module that uses the UMS Enterprise JavaBeans Client API:

- Enterprise JavaBeans Application Module - Stateless Session Bean - This is a back end, core message-receiving or message-sending application.
- Web Application Module - This is for applications that have an HTML or web front end.

Whichever application module is selected uses the UMS Client API to register the application with the UMS Server and subsequently invoke operations to send or retrieve messages, status, and register or unregister access points. For a complete list of operations refer to the UMS Javadoc.

The samples with source code are available on Oracle Technology Network (OTN).

60.2 Creating a UMS Client Instance

This section describes the requirements for creating a UMS Enterprise JavaBeans Client. You can create a `MessagingEJBClient` instance by using the code in the MessagingClientFactory class.

When creating an application using the UMS Enterprise JavaBeans Client, the application must be packaged as an EAR file, and the `usermessagingclient-ejb.jar` module bundled as an Enterprise JavaBeans module.

60.2.1 Creating a MessagingEJBClient Instance Using a Programmatic or Declarative Approach

Example 60–1 shows code for creating a `MessagingEJBClient` instance using the programmatic approach:

Example 60–1 Programmatic Approach to Creating a MessagingEJBClient Instance

```
ApplicationInfo appInfo = new ApplicationInfo();
appInfo.setApplicationName("SampleApp");
appInfo.setApplicationInstanceName("SampleAppInstance");
MessagingClient mClient =
 MessagingClientFactory.createMessagingEJBClient(appInfo);
```

You can also create a `MessagingEJBClient` instance using a declarative approach. The declarative approach is normally the preferred approach since it enables you to make changes at deployment time.

You must specify all the required Application Info properties as environment entries in your Java EE module's descriptor (`ejb-jar.xml` or `web.xml`).

Example 60–2 shows code for creating a `MessagingEJBClient` instance using the declarative approach:

Example 60–2 Declarative Approach to Creating a MessagingEJBClient Instance

```
MessagingClient mClient = MessagingClientFactory.createMessagingEJBClient();
```

60.2.2 API Reference for Class MessagingClientFactory

The API reference for class `MessagingClientFactory` can be accessed from the Javadoc.

60.3 Sending a Message

You can create a message by using the code in the `MessageFactory` class and `Message` interface of `oracle.sdp.messaging`.

The types of messages that can be created include plaintext messages, multipart messages that can consist of text/plain and text/html parts, and messages that include the creation of delivery channel (`DeliveryType`) specific payloads in a single message for recipients with different delivery types.

60.3.1 Creating a Message

This section describes the various types of messages that can be created.

60.3.1.1 Creating a Plaintext Message

Example 60–3 shows how to create a plain text message using the UMS Java API.

Example 60–3 Creating a Plaintext Message Using the UMS Java API

```
Message message = MessageFactory.getInstance().createTextMessage("This is a Plain
Text message.");
Message message = MessageFactory.getInstance().createMessage();
message.setContent("This is a Plain Text message.", "text/plain");
```

60.3.1.2 Creating a Multipart/Alternative Message (with Text/Plain and Text/HTML Parts)

Example 60–4 shows how to create a multipart or alternative message using the UMS Java API.

Example 60–4 Creating a Multipart or Alternative Message Using the UMS Java API

```
Message message = MessageFactory.getInstance().createMessage();
MimeMultipart mp = new MimeMultipart("alternative");
MimeBodyPart mp_partPlain = new MimeBodyPart();
mp_partPlain.setContent("This is a Plain Text part.", "text/plain");
mp.addBodyPart(mp_partPlain);
MimeBodyPart mp_partRich = new MimeBodyPart();
mp_partRich
        .setContent(
                "<html><head></head><body><b><i>This is an HTML
part.</i></b></body></html>",
                "text/html");
mp.addBodyPart(mp_partRich);
message.setContent(mp, "multipart/alternative");
```

60.3.1.3 Creating Delivery Channel-Specific Payloads in a Single Message for Recipients with Different Delivery Types

When sending a message to a destination address, there can be multiple channels involved. Oracle UMS application developers are required to specify the correct multipart format for each channel.

Example 60–5 shows how to create delivery channel (`DeliveryType`) specific payloads in a single message for recipients with different delivery types.

Each top-level part of a multiple payload multipart/alternative message should contain one or more values of this header. The value of this header should be the name of a valid delivery type. Refer to the available values for *DeliveryType* in the enum DeliveryType.

Example 60–5 Creating Delivery Channel-specific Payloads in a Single Message for Recipients with Different Delivery Types

```
Message message = MessageFactory.getInstance().createMessage();

// create a top-level multipart/alternative MimeMultipart object.
MimeMultipart mp = new MimeMultipart("alternative");

// create first part for SMS payload content.
MimeBodyPart part1 = new MimeBodyPart();
part1.setContent("Text content for SMS.", "text/plain");

part1.setHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "SMS");

// add first part
mp.addBodyPart(part1);

// create second part for EMAIL and IM payload content.
MimeBodyPart part2 = new MimeBodyPart();
MimeMultipart part2_mp = new MimeMultipart("alternative");
MimeBodyPart part2_mp_partPlain = new MimeBodyPart();
part2_mp_partPlain.setContent("Text content for EMAIL/IM.", "text/plain");
part2_mp.addBodyPart(part2_mp_partPlain);
MimeBodyPart part2_mp_partRich = new MimeBodyPart();
part2_mp_partRich.setContent("<html><head></head><body><b><i>" + "HTML content for
EMAIL/IM." +
"</i></b></body></html>", "text/html");
part2_mp.addBodyPart(part2_mp_partRich);
part2.setContent(part2_mp, "multipart/alternative");

part2.addHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "EMAIL");
part2.addHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "IM");

// add second part
mp.addBodyPart(part2);

// set the content of the message
message.setContent(mp, "multipart/alternative");

// set the MultiplePayload flag to true
message.setMultiplePayload(true);
```

60.3.2 API Reference for Class MessageFactory

The API reference for class `MessageFactory` can be accessed from the Javadoc.

60.3.3 API Reference for Interface Message

The API reference for interface `Message` can be accessed from the Javadoc.

60.3.4 API Reference for Enum DeliveryType

The API reference for enum `DeliveryType` can be accessed from the Javadoc.

60.3.5 Addressing a Message

This section describes type of addresses and how to create address objects.

60.3.5.1 Types of Addresses

There are two types of addresses, *device addresses* and *user addresses*. A device address can be of various types, such as email addresses, instant messaging addresses, and telephone numbers. User addresses are user IDs in a user repository.

60.3.5.2 Creating Address Objects

You can address senders and recipients of messages by using the class `AddressFactory` to create `Address` objects defined by the `Address` interface.

60.3.5.2.1 Creating a Single Address Object Example 60–6 shows code for creating a single `Address` object:

Example 60–6 Creating a Single Address Object

```
Address recipient =
AddressFactory.getInstance().createAddress("Email:john.doe@oracle.com");
```

60.3.5.2.2 Creating Multiple Address Objects in a Batch Example 60–7 shows code for creating multiple `Address` objects in a batch:

Example 60–7 Creating Multiple Address Objects in a Batch

```
String[] recipientsStr = {"Email:john.doe@oracle.com",
"IM:jabber|john.doe@oracle.com"};
Address[] recipients = AddressFactory.getInstance().createAddress(recipientsStr);
```

60.3.5.2.3 Adding Sender or Recipient Addresses to a Message Example 60–8 shows code for adding sender or recipient addresses to a message:

Example 60–8 Adding Sender or Recipient Addresses to a Message

```
Address sender =
AddressFactory.getInstance().createAddress("Email:john.doe@oracle.com");
Address recipient =
AddressFactory.getInstance().createAddress("Email:jane.doe@oracle.com");
message.addSender(sender);
message.addRecipient(recipient);
```

60.3.5.3 Creating a Recipient with a Failover Address

Example 60–9 shows code for creating a recipient with a failover address:

Example 60–9 Creating a Single Address Object with Failover

```
String recipientWithFailoverStr = "Email:john.doe@oracle.com,
IM:jabber|john.doe@oracle.com";
```

```
Address recipient =
AddressFactory.getInstance().createAddress(recipientWithFailoverStr);
```

60.3.5.4 API Reference for Class AddressFactory

The API reference for class `AddressFactory` can be accessed from the Javadoc.

60.3.5.5 API Reference for Interface Address

The API reference for interface `Address` can be accessed from the Javadoc.

60.3.6 Retrieving Message Status

You can use Oracle UMS to retrieve message status either synchronously or asynchronously.

60.3.6.1 Synchronous Retrieval of Message Status

To perform a synchronous retrieval of current status, use the following flow from the `MessagingClient` API:

```
String messageId = messagingClient.send(message);
Status[] statuses = messagingClient.getStatus(messageId);
```

or,

```
Status[] statuses = messagingClient.getStatus(messageId, address[]) --- where
 address[] is an array of one or more of the recipients set in the message.
```

60.3.6.2 Asynchronous Notification of Message Status

To retrieve an asynchronous notification of message status, perform the following:

1. Implement a status listener.
2. Register a status listener (declarative way)
3. Send a message (`messagingClient.send(message);`)
4. The application automatically gets the status through an `onStatus(status)` callback of the status listener.

60.4 Receiving a Message

This section describes how an application receives messages. To receive a message you must first register an access point. From the application perspective there are two modes for receiving a message, synchronous and asynchronous.

60.4.1 Registering an Access Point

`AccessPoint` represents one or more device addresses to receive incoming messages. An application that wants to receive incoming messages must register one or more access points that represent the recipient addresses of the messages. The server matches the recipient address of an incoming message against the set of registered access points, and routes the incoming message to the application that registered the matching access point.

You can use `AccessPointFactory.createAccessPoint` to create an access point and `MessagingClient.registerAccessPoint` to register it for receiving messages.

To register an SMS access point for the number `9000`:

```
AccessPoint accessPointSingleAddress =
 AccessPointFactory.createAccessPoint(AccessPoint.AccessPointType.SINGLE_ADDRESS,
 DeliveryType.SMS, "9000");
messagingClient.registerAccessPoint(accessPointSingleAddress);
```

To register SMS access points in the number range `9000` to `9999`:

```
AccessPoint accessPointRangeAddress =
 AccessPointFactory.createAccessPoint(AccessPoint.AccessPointType.NUMBER_RANGE,
 DeliveryType.SMS,"9000,9999");
messagingClient.registerAccessPoint(accessPointRangeAddress);
```

60.4.2 Synchronous Receiving

You can use the method `MessagingClient.receive` to synchronously receive messages. This is a convenient polling method for light-weight clients that do not want the configuration overhead associated with receiving messages asynchronously. This method returns a list of messages that are immediately available in the application inbound queue.

It performs a nonblocking call, so if no message is currently available, the method returns null.

Note: A single invocation does not guarantee retrieval of all available messages. You must poll to ensure receiving all available messages.

60.4.3 Asynchronous Receiving

Asynchronous receiving involves many tasks, including configuring MDBs and writing a Stateless Session Bean message listener. See the sample application usermessagingsample-echo for detailed instructions.

60.4.4 Message Filtering

A `MessageFilter` is used by an application to exercise greater control over what messages are delivered to it. A `MessageFilter` contains a matching criterion and an action. An application can register a series of message filters; they are applied in order against an incoming (received) message; if the criterion matches the message, the action is taken. For example, an application can use `MessageFilters` to implement necessary blacklists, by rejecting all messages from a given sender address.

You can use `MessageFilterFactory.createMessageFilter` to create a message filter, and `MessagingClient.registerMessageFilter` to register it. The filter is added to the end of the current filter chain for the application. When a message is received, it is passed through the filter chain in order; if the message matches a filter's criterion, the filter's action is taken immediately. If no filters match the message, the default action is to accept the message and deliver it to the application.

For example, to reject a message with the subject `"spam"`:

```
MessageFilter subjectFilter = MessageFilterFactory.createMessageFilter("spam",
 MessageFilter.FieldType.SUBJECT, null, MessageFilter.Action.REJECT);
messagingClient.registerMessageFilter(subjectFilter);
```

To reject messages from email address `spammer@foo.com`:

```
MessageFilter senderFilter =
 MessageFilterFactory.createBlacklistFilter("spammer@foo.com");
messagingClient.registerMessageFilter(senderFilter);
```

60.5 Using the UMS Enterprise JavaBeans Client API to Build a Client Application

This section describes how to create an application called *usermessagingsample*, a web client application that uses the UMS Enterprise JavaBeans Client API for both outbound messaging and the synchronous retrieval of message status. *usermessagingsample* also supports inbound messaging. Once you have deployed and configured *usermessagingsample*, you can use it to send a message to an email client.

Note: To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL: `https://soasamples.samplecode.oracle.com/`

Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

Of the two application modules choices described in Section 60.1.1, "Creating a Java EE Application Module," this sample focuses on the Web Application Module (WAR), which defines some HTML forms and servlets. You can examine the code and corresponding XML files for the web application module from the provided `usermessagingsample-src.zip` source. The servlets uses the UMS Enterprise JavaBeans Client API to create an UMS Enterprise JavaBeans Client instance (which in turn registers the application's info) and sends messages.

This application, which is packaged as an Enterprise Archive file (EAR) called *usermessagingsample-ejb.ear*, has the following structure:

- `usermessagingsample-ejb.ear`
  - `META-INF`
    - `application.xml` -- Descriptor file for all of the application modules.
    - `weblogic-application.xml` -- Descriptor file that contains the `import` of the `oracle.sdp.messaging` shared library.
  - `usermessagingclient-ejb.jar` -- Contains the Message Enterprise JavaBeans Client deployment descriptors.
    - `META-INF`
      - `ejb-jar.xml`
      - `weblogic-ejb-jar.xml`
  - `usermessagingsample-web.ear` -- Contains the web-based front-end and servlets.
    - `WEB-INF`
      - `web.xml`
      - `weblogic.xml`

The prebuilt sample application, and the source code (usermessagingsample-src.zip) are available on OTN.

60.5.1 Overview of Development

The following steps describe the process of building an application capable of outbound messaging using `usermessagingsample-ejb.ear` as an example:

1. Section 60.5.2, "Configuring the Email Driver"
2. Section 60.5.3, "Using JDeveloper 11g to Build the Application"
3. Section 60.5.4, "Deploying the Application"
4. Section 60.5.5, "Testing the Application"

60.5.2 Configuring the Email Driver

To enable the Oracle User Messaging Service's email driver to perform outbound messaging and status retrieval, configure the email driver as follows:

- Enter the name of the SMTP mail server as the value for the `OutgoingMailServer` property.

> **Note:** This sample application is generic and can support outbound messaging through other channels when the appropriate messaging drivers are deployed and configured.

60.5.3 Using JDeveloper 11g to Build the Application

This section describes using a Windows-based build of JDeveloper to build, compile, and deploy *usermessagingsample* through the following steps:

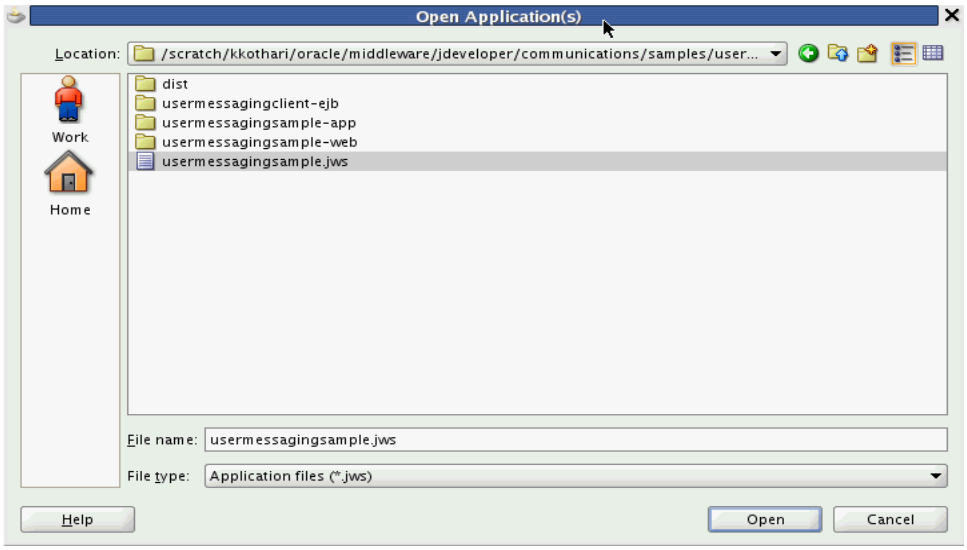
60.5.3.1 Opening the Project

1. Unzip `usermessagingsample-src.zip`, to the *`JDEV_HOME`*`/communications/samples/` directory. This directory must be used for the shared library references to be valid in the project.

> **Note:** If you choose to use a different directory, you must update the `oracle.sdp.messaging` library source path to *`JDEV_HOME`*`/communications/modules/oracle.sdp.messaging_11.1.1/sdpmessaging.jar`.

2. Open `usermessagingsample.jws` (contained in the .zip file) in Oracle JDeveloper.

Figure 60–1 Oracle JDeveloper Main Window



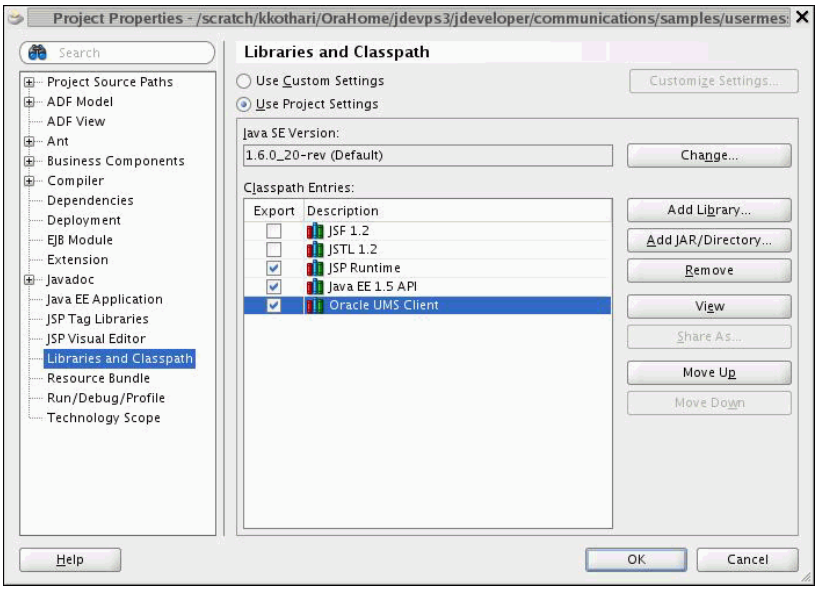
In the Oracle JDeveloper main window, the project appears.

3. Satisfy the build dependencies for the sample application by ensuring the "Oracle UMS Client" library is used by the Web module.

    1. In the Application Navigator, right-click web module **usermessagingsample-web**, and select **Project Properties**.

    2. In the left pane, select **Libraries and Classpath**.

Figure 60–2 Verifying Libraries



    3. Click **OK**.

4. Verify that the **usermessagingclient-ejb** project exists in the application. This is an Enterprise JavaBeans module that packages the messaging client beans used by UMS applications. The module allows the application to connect with the UMS server.

5. Explore the Java files under the **usermessagingsample-web** project to see how the messaging client APIs are used to send messages, get statuses, and synchronously receive messages. The application info that is registered with the UMS Server is specified programmatically in `SampleUtils.java` in the project (Example 60–10).

Example 60–10 Application Information

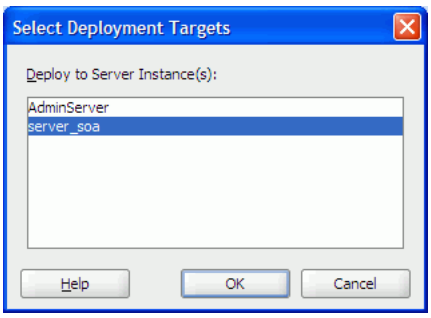
```
ApplicationInfo appInfo = new ApplicationInfo();
appInfo.setApplicationName(SampleConstants.APP_NAME);
appInfo.setApplicationInstanceName(SampleConstants.APP_INSTANCE_NAME);
appInfo.setSecurityPrincipal(request.getUserPrincipal().getName());
```

60.5.4 Deploying the Application

Perform the following steps to deploy the application:

1. Create an Application Server Connection by right-clicking the application in the navigation pane and selecting New. Follow the instructions in Section 60.7, "Creating a New Application Server Connection."
2. Deploy the application by selecting the **usermessagingsample application**, **Deploy**, **usermessagingsample**, **to**, and **SOA_server** (Figure 60–3).

Figure 60–3 Deploying the Project



3. Verify that the message `Build Successful` appears in the log.
4. Verify that the message `Deployment Finished` appears in the deployment log.

   You have successfully deployed the application.

   Before you can run the sample, you must configure any additional drivers in Oracle User Messaging Service and optionally configure a default device for the user receiving the message in User Messaging Preferences.

> **Note:** Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

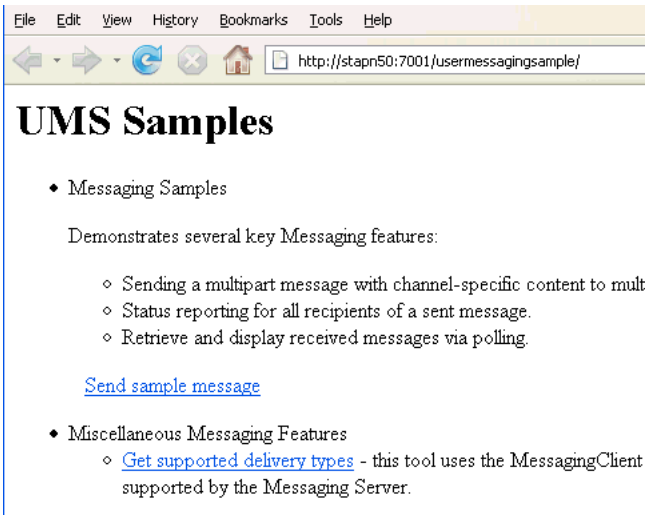
60.5.5 Testing the Application

Once **usermessagingsample** has been deployed to a running instance of Oracle WebLogic Server, perform the following:

1. Launch a web browser and enter the address of the sample application as follows: `http://`*`host`*`:`*`http-port`*`/usermessagingsample/`. For example, enter `http://localhost:7001/usermessagingsample/` into the browser's navigation bar.

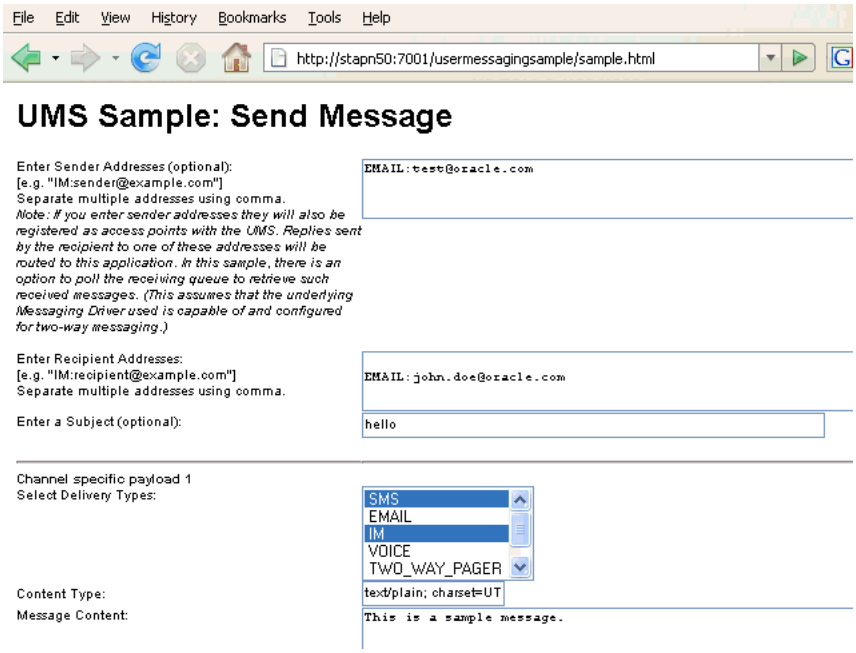
When prompted, enter login credentials. For example, username `weblogic`. The browser page for testing messaging samples appears (Figure 60–4).

Figure 60–4 Testing the Sample Application



2. Click **Send sample message**. The Send Message page appears (Figure 60–5).

Figure 60–5 Addressing the Test Message



3. As an optional step, enter the sender address in the following format:

   `Email:`*`sender_address`*.

   For example, enter `Email:sender@oracle.com`.

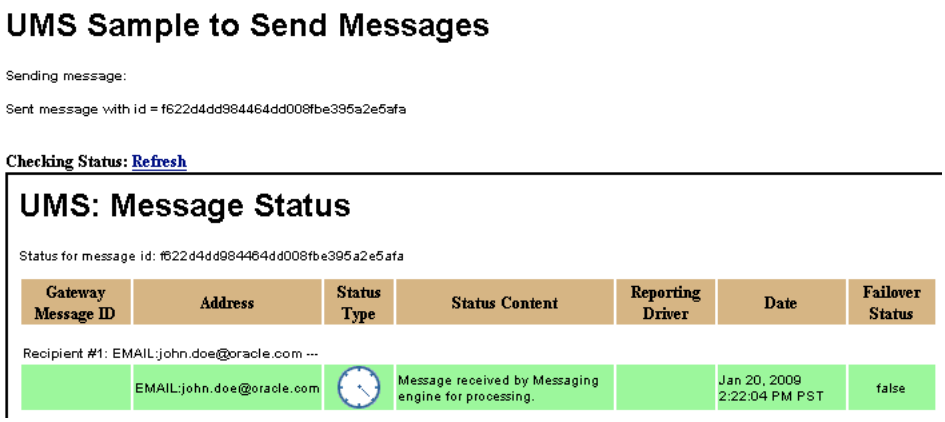
4. Enter one or more recipient addresses. For example, enter `Email:recipient@oracle.com`. Enter multiple addresses as a comma-separated list as follows:

`Email:`*`recipient_address1`*`, Email:`*`recipient_address2`*.

If you have configured user messaging preferences, you can address the message simply to `User:`*`username`*. For example, `User:weblogic`.

5. As an optional step, enter a subject line or content for the email.
6. Click **Send**. The Message Status page appears, showing the progress of transaction (Message received by Messaging engine for processing in Figure 60–6).

Figure 60–6 Message Status



7. Click **Refresh** to update the status. When the email message has been delivered to the email server, the *Status Content* field displays *Outbound message delivery to remote gateway succeeded*.

60.6 Using the UMS Enterprise JavaBeans Client API to Build a Client Echo Application

This section describes how to create an application called usermessagingsample-echo, a demo client application that uses the UMS Enterprise JavaBeans Client API to asynchronously receive messages from an email address and echo a reply back to the sender.

> **Note:** To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL:
> `https://soasamples.samplecode.oracle.com/`
>
> Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

This application, which is packaged as a Enterprise Archive file (EAR) called *usermessagingsample-echo-ejb.ear*, has the following structure:

- `usermessagingsample-echo-ejb.ear`
  - `META-INF`
    - `application.xml` -- Descriptor file for all of the application modules.

  - `weblogic-application.xml` -- Descriptor file that contains the `import` of the `oracle.sdp.messaging` shared library.

- `usermessagingclient-ejb.jar` -- Contains the Message Enterprise JavaBeans Client deployment descriptors.
  * `META-INF`
  - `ejb-jar.xml`
  - `weblogic-ejb-jar.xml`
- `usermessagingsample-echo-ejb.jar` -- Contains the application session beans (ClientSenderBean, ClientReceiverBean) that process a received message and return an echo response.
  * `META-INF`
  - `ejb-jar.xml`
  - `weblogic-ejb-jar.xml`
- `usermessagingsample-echo-web.war` -- Contains the web-based front-end and servlets.
  * `WEB-INF`
  - `web.xml`
  - `weblogic.xml`

The prebuilt sample application, and the source code (`usermessagingsample-echo-src.zip`) are available on OTN.

60.6.1 Overview of Development

The following steps describe the process of building an application capable of asynchronous inbound and outbound messaging using `usermessagingsample-echo-ejb.ear` as an example:

1. Section 60.6.2, "Configuring the Email Driver"
2. Section 60.6.3, "Using JDeveloper 11g to Build the Application"
3. Section 60.6.4, "Deploying the Application"
4. Section 60.6.5, "Testing the Application"

60.6.2 Configuring the Email Driver

To enable the Oracle User Messaging Service's email driver to perform inbound and outbound messaging and status retrieval, configure the email driver as follows:

- Enter the name of the SMTP mail server as the value for the **OutgoingMailServer** property.
- Enter the name of the IMAP4/POP3 mail server as the value for the **IncomingMailServer** property. Also, configure the incoming user name, and password.

> **Note:** This sample application is generic and can support inbound and outbound messaging through other channels when the appropriate messaging drivers are deployed and configured.

60.6.3 Using JDeveloper 11g to Build the Application

This section describes using a Windows-based build of JDeveloper to build, compile, and deploy **usermessagingsample-echo** through the following steps:

60.6.3.1 Opening the Project

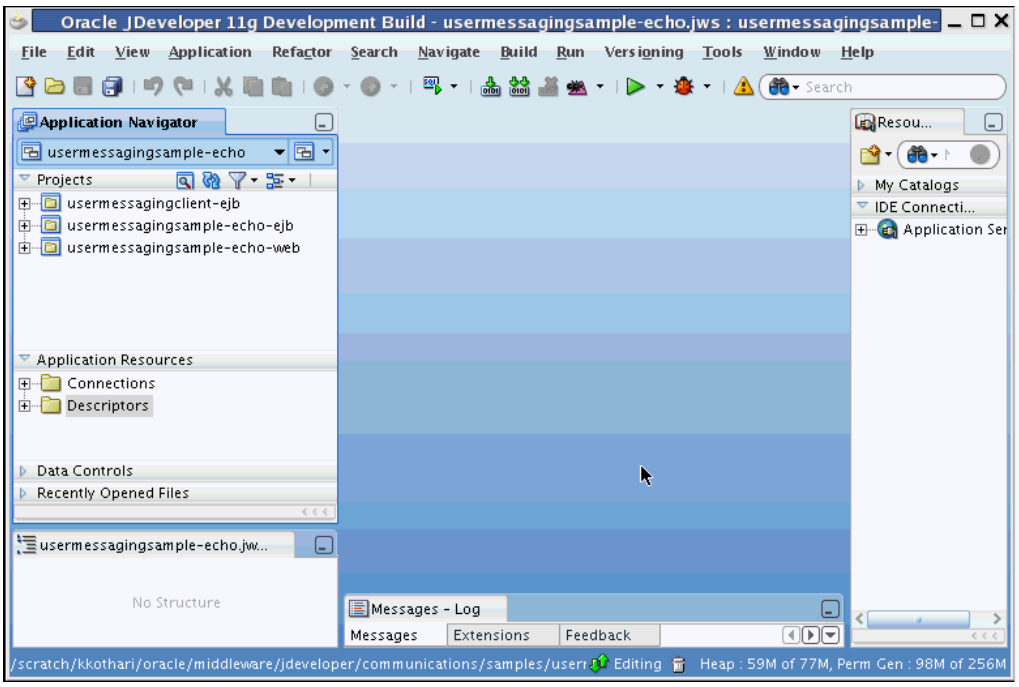
1. Unzip `usermessagingsample.echo-src.zip`, to the *JDEV_HOME*`/communications/samples/` directory. This directory must be used for the shared library references to be valid in the project.

   Note: If you choose to use a different directory, you must update the oracle.sdp.messaging library source path to *JDEV_HOME*`/communications/modules/oracle.sdp.messaging_11.1.1/sdpmessaging.jar`.

2. Open `usermessagingsample-echo.jws` (contained in the .zip file) in Oracle JDeveloper.

   In the Oracle JDeveloper main window, the project appears (Figure 60–7).

Figure 60–7 Oracle JDeveloper Main Window



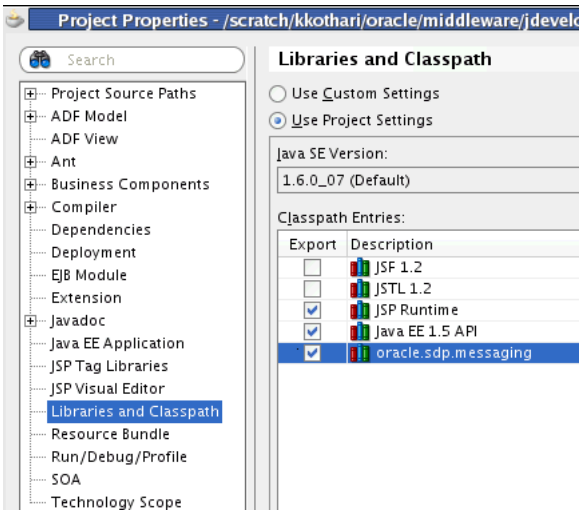
3. Verify that the build dependencies for the sample application have been satisfied by checking that the following library has been added to the `usermessagingsample-echo-web` and `usermessagingsample-echo-ejb` modules.

   - Library: *oracle.sdp.messaging*, Classpath: *JDEV_HOME*`/communications/modules/oracle.sdp.messaging_11.1.1/sdpmessaging.jar`. This is the Java library used by UMS and applications that use UMS to send and receive messages.

   Perform the following steps for each module:

1. In the Application Navigator, right-click the module and select **Project Properties**.
2. In the left pane, select **Libraries and Classpath** (Figure 60–8).

Figure 60–8 Verifying Libraries



3. Click **OK**.

4. Verify that the **usermessagingclient-ejb** project exists in the application. This is an Enterprise JavaBeans module that packages the messaging client beans used by UMS applications. The module allows the application to connect with the UMS server.
5. Explore the Java files under the **usermessagingsample-echo-ejb** project to see how the messaging client APIs are used to asynchronously receive messages (`ClientReceiverBean`), and send messages (`ClientSenderBean`).
6. Explore the Java files under the **usermessagingsample-echo-web** project to see how the messaging client APIs are used to register and unregister access points.
7. Note that the application info that is registered with the UMS Server is specified declaratively in the **usermessagingclient-ejb** project's `ejb-jar.xml` file. (Example 60–11).

Example 60–11 Application Information

```
            <env-entry>
                <env-entry-name>sdpm/ApplicationName</env-entry-name>
                <env-entry-type>java.lang.String</env-entry-type>
                <env-entry-value>UMSEchoApp</env-entry-value>
            </env-entry>
            <env-entry>
                <env-entry-name>sdpm/ApplicationInstanceName</env-entry-name>
                <env-entry-type>java.lang.String</env-entry-type>
                <env-entry-value>UMSEchoAppInstance</env-entry-value>
            </env-entry>

            <env-entry>
               <env-entry-name>sdpm/ReceivingQueuesInfo</env-entry-name>
                <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>OraSDPM/QueueConnectionFactory:OraSDPM/Queues/OraSDPMAppDefRcvQ1<
```

```
/env-entry-value>
            </env-entry>

            <env-entry>
               <env-entry-name>
               sdpm/MessageListenerSessionBeanJNDIName
                </env-entry-name>
                <env-entry-type>java.lang.String</env-entry-type>
                <env-entry-value>
                  ejb/umsEchoApp/ClientReceiverLocal</env-entry-value>
            </env-entry>
            <env-entry>
                <env-entry-name>
                sdpm/MessageListenerSessionBeanHomeClassName</env-entry-name>
                <env-entry-type>java.lang.String</env-entry-type>
                <env-entry-value>
                oracle.sdp.messaging.sample.ejbApp.ClientReceiverHomeLocal
                 </env-entry-value>
              </env-entry>
              <env-entry>
                <env-entry-name>
                sdpm/StatusListenerSessionBeanJNDIName
               </env-entry-name>
                <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>ejb/umsEchoApp/ClientReceiverLocal</env-entry-value>
            </env-entry>
            <env-entry>

<env-entry-name>sdpm/StatusListenerSessionBeanHomeClassName</env-entry-name>
                <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>oracle.sdp.messaging.sample.ejbApp.ClientReceiverHomeLocal</env-e
ntry-value>
            </env-entry>
```

8. Note that the Application Name (`UMSEchoApp`) and Application Instance Name (`UMSEchoAppInstance`) are also used in the Message Selector for the `MessageDispatcherBean` MDB, which is used for asynchronous receiving of messages and statuses placed in the application receiving queue (Example 60–12).

Example 60–12 Application Information

```
<activation-config-property>
  <activation-config-property-name>
    messageSelector
  </activation-config-property-name>
  <activation-config-property-value>
    appName='UMSEchoApp' or sessionName='UMSEchoApp-UMSEchoAppInstance'
  </activation-config-property-value>
</activation-config-property>
```

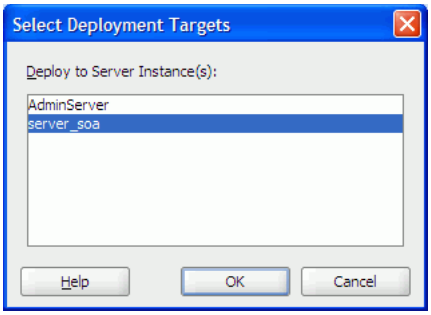
Note: If you chose a different Application Name and Application Instance Name for your own application, remember to update this message selector. Asynchronous receiving does not work, otherwise.

60.6.4 Deploying the Application

Perform the following steps to deploy the application:

1. Create an Application Server Connection by right-clicking the application in the navigation pane and selecting New. Follow the instructions in Section 60.7, "Creating a New Application Server Connection."
2. Deploy the application by selecting the **usermessagingsample-echo application**, **Deploy**, **usermessagingsample-echo**, **to**, and **SOA_server** (Figure 60–9).

Figure 60–9 Deploying the Project



3. Verify that the message `Build Successful` appears in the log.
4. Verify that the message `Deployment Finished` appears in the deployment log.

   You have successfully deployed the application.

   Before you can run the sample you must configure any additional drivers in Oracle User Messaging Service and optionally configure a default device for the user receiving the message in User Messaging Preferences.

Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

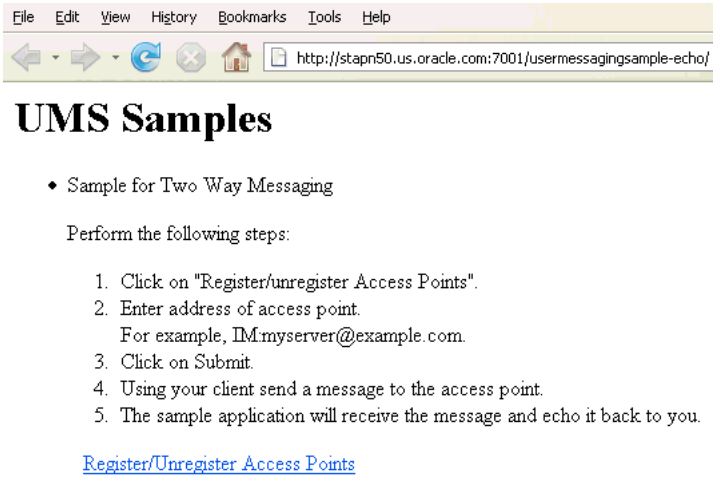
60.6.5 Testing the Application

Once **usermessagingsample-echo** has been deployed to a running instance of Oracle WebLogic Server, perform the following:

1. Launch a web browser and enter the address of the sample application as follows: `http://`*`host`*`:`*`http-port`*`/usermessagingsample-echo/`. For example, enter `http://localhost:7001/usermessagingsample-echo/` into the browser's navigation bar.

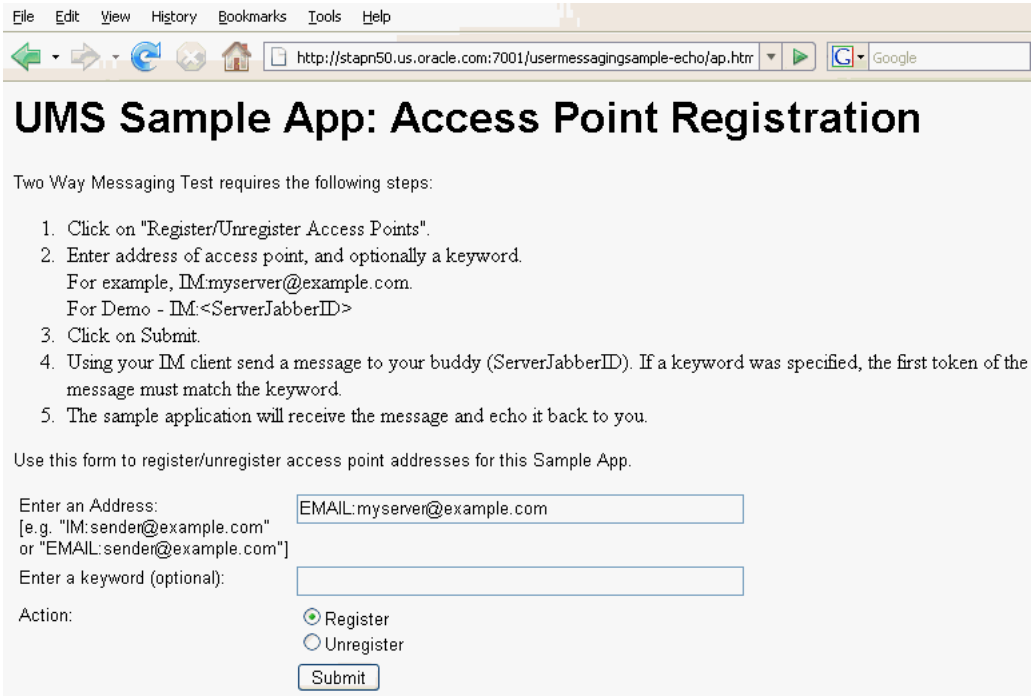
   When prompted, enter login credentials. For example, username `weblogic`. The browser page for testing messaging samples appears (Figure 60–10).

Figure 60–10 Testing the Sample Application



2. Click **Register/Unregister Access Points**. The *Access Point Registration* page appears (Figure 60–11).

Figure 60–11 Registering an Access Point



3. Enter the access point address in the following format:

   `EMAIL:server_address`.

   For example, enter `EMAIL:myserver@example.com`.

4. Select the Action **Register** and Click **Submit**. The registration status page appears, showing "Registered" in Figure 60–12).

Figure 60–12 Access Point Registration Status

5. Send a message from your messaging client (for email, your email client) to the address you just registered as an access point in the previous step.

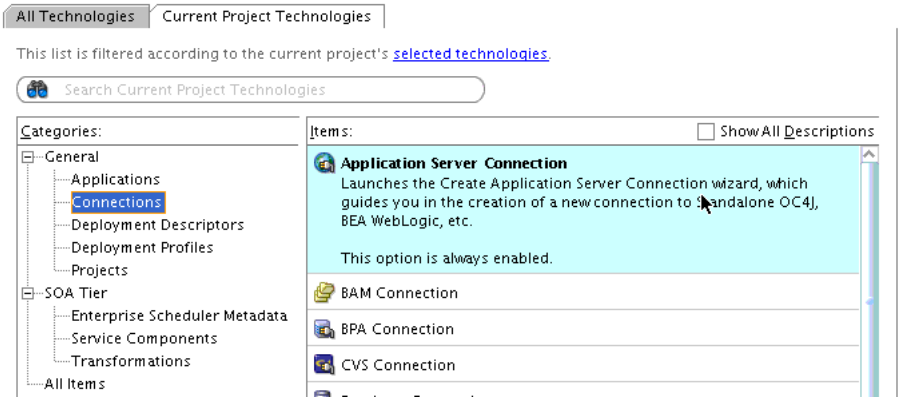
   If the UMS messaging driver for that channel is configured correctly, you should expect to receive an echo message back from the **usermessagingsample-echo** application.

60.7 Creating a New Application Server Connection

Perform the following steps to create an Application Server Connection.

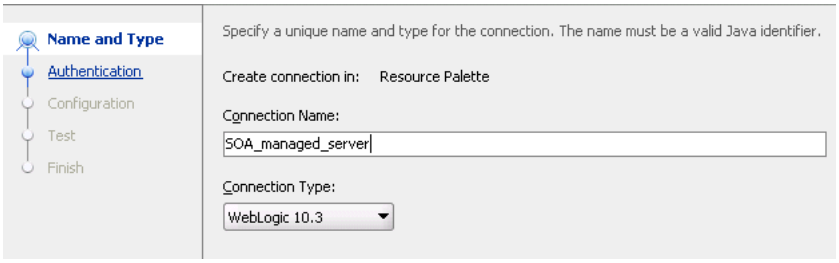
1. Create a new Application Server Connection by right-clicking the project and selecting **New**, **Connections**, and **Application Server Connection** (Figure 60–13).

Figure 60–13 New Application Server Connection



2. Name the connection `SOA_server` and click **Next** (Figure 60–14).
3. Select **WebLogic 10.3** as the **Connection Type**.

Figure 60–14 New Application Server Connection



4. Enter the authentication information. A typical value for user name is `weblogic`.
5. In the Connection dialog, enter the hostname, port, and SSL port for the SOA admin server, and enter the name of the domain for WLS Domain.
6. Click **Next**.
7. In the Test dialog, click **Test Connection**.
8. Verify that the message `Success!` appears.

   The Application Server Connection has been created.

61

Sending and Receiving Messages using the User Messaging Service Java API

This chapter describes how to use the User Messaging Service (UMS) client API to develop applications. This API serves as a programmatic entry point for Fusion Middleware application developers to incorporate messaging features within their enterprise applications.

Because the API provides a plain old java (POJO/POJI) programming model, this eliminates the needs for application developers to package and implement various Java EE modules (such as an EJB module) in an application to access UMS features. This reduces application development time because developers can create applications to run in a Java EE container without performing any additional packaging of modules, or obtaining specialized tools to perform such packaging tasks.

Consumers of the UMS Java API are not required to use any Java EE mechanism such as environment entries or other Java EE deployment descriptor artifacts. Besides the overhead involved in maintaining Java EE descriptors, many client applications already have a configuration framework that does not rely on Java EE descriptors.

This chapter includes the following sections:

Note: To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL: `https://soasamples.samplecode.oracle.com/`

Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

61.1 Introduction to the UMS Java API

The UMS Java API is exposed as a POJO/POJI API. Consumers of the API can get an instance of a `MessagingClient` object using a factory method. The consumers do not need to deploy any EJB or other Java EE modules in their applications, but must ensure that the UMS libraries are available in an application' s runtime class path. The deployment is as a shared library, "oracle.sdp.messaging".

The UMS Java API consists of packages grouped as follows:

- Common and Client Packages
  - `oracle.sdp.messaging`
  - `oracle.sdp.messaging.filter`: A MessageFilter is used by an application to exercise greater control over what messages are delivered to it.

The samples with source code are available on Oracle Technology Network (OTN).

61.2 Creating a UMS Client Instance and Specifying Runtime Parameters

This section describes the requirements for creating a UMS Client. You can create a `MessagingClient` instance by using the code in the MessagingClientFactory class. Specifically, use the `MessagingClientFactory.createMessagingClient()` method to create the instance.

Client applications can specify a set of parameters at runtime when instantiating a client object. For example, you configure a `MessagingClient` instance by specifying parameters as a map of key-value pairs in a `java.util.Map<String, Object>`. Among other things, the configuration parameters serve to identify the client application, point to the UMS server, and establish security credentials. Client applications are responsible for storing and loading the configuration parameters using any available mechanism.

Table 61–1 lists some configuration parameters that may be set for the Java API. In typical use cases, most of the parameters do not need to be provided and the API implementation uses sensible default values.

Table 61–1 Configuration Parameters Specified at Runtime

Parameter	Notes
`APPLICATION_NAME`	Optional. By default, the client is identified by its deployment name. This identifier can be overridden by specifying a value for key `ApplicationInfo.APPLICATION_NAME`.
`APPLICATION_INSTANCE_NAME`	Optional. Only required for certain clustered use cases or to take advantage of session-based routing.

Table 61–1 (Cont.) Configuration Parameters Specified at Runtime

Parameter	Notes
SDPM_SECURITY_PRINCIPAL	Optional. By default, the client's resources are available to any application with the same application name and any security principal. This behavior can be overridden by specifying a value for key ApplicationInfo.SDPM_SECURITY_PRINCIPAL. If a security principal is specified, then all subsequent requests involving the application's resources (messages, access points, and so on.) must be made using the same security principal.
MESSAGE_LISTENER_THREADS STATUS_LISTENER_THREADS	Optional. When listeners are used to receive messages or statuses asynchronously, the number of listener worker threads can be controlled by specifying values for the MessagingConstants.MESSAGE_LISTENER_THREADS and MessagingConstants.STATUS_LISTENER_THREADS keys.
RECEIVE_ACKNOWLEDGEMENT_MODE LISTENER_ACKNOWLEDGEMENT_MODE	Optional. When receiving messages, you can control the reliability mode by specifying values for the MessagingConstants.RECEIVE_ACKNOWLEDGEMENT_MODE (synchronous receiving) and MessagingConstants.LISTENER_ACKNOWLEDGEMENT_MODE (asynchronous receiving) keys.

A MessagingClient cannot be reconfigured after it is instantiated. Instead, a new instance of the MessagingClient class must be created using the new configuration.

To release resources used by the MessagingClient instance when it is no longer needed, call MessagingClientFactory.remove(client). If you do not call this method, some resources such as worker threads and JMS listeners may remain active.

Example 61–1 shows code for creating a MessagingClient instance using the programmatic approach:

Example 61–1 Programmatic Approach to Creating a MessagingClient Instance

```
Map<String, Object> params = new HashMap<String, Object>();
// params.put(key, value);  // if optional parameters need to be specified.
MessagingClient messagingClient =
MessagingClientFactory.createMessagingClient(params);
```

A MessagingClient cannot be reconfigured after it is instantiated. Instead, you must create a new instance of the MessagingClient class using the desired configuration.

61.2.1 API Reference for Class MessagingClientFactory

The API reference for class MessagingClientFactory can be accessed from the Javadoc.

61.3 Sending a Message

The client application can create a message object using the `MessagingFactory` class of `oracle.sdp.messaging`. `MessagingFactory` is a factory class to create various messaging objects. (You can use other methods in this class to create `Addresses`, `AccessPoints`, `MessageFilters`, and `MessageQueries`. See the Javadoc for these methods).

The client application can then send the message. The API returns a String identifier that the client application can later use to retrieve message delivery status. The status returned is the latest known status based on UMS internal processing and delivery notifications received from external gateways.

The types of messages that can be created include plaintext messages, multipart messages that can consist of text/plain and text/html parts, and messages that include the creation of delivery channel (`DeliveryType`) specific payloads in a single message for recipients with different delivery types.

61.3.1 Creating a Message

This section describes the various types of messages that can be created.

61.3.1.1 Creating a Plaintext Message

Example 61–2 shows how to create a plaintext message using the UMS Java API.

Example 61–2 Creating a Plaintext Message Using the UMS Java API

```
Message message = MessagingFactory.createTextMessage("This is a Plain Text
message.");
Message message = MessagingFactory.createMessage();
message.setContent("This is a Plain Text message.", "text/plain");
```

61.3.1.2 Creating a Multipart/Alternative Message (with Text/Plain and Text/HTML Parts)

Example 61–3 shows how to create a multipart or alternative message using the UMS Java API.

Example 61–3 Creating a Multipart or Alternative Message Using the UMS Java API

```
Message message = MessagingFactory.createMessage();
MimeMultipart mp = new MimeMultipart("alternative");
MimeBodyPart mp_partPlain = new MimeBodyPart();
mp_partPlain.setContent("This is a Plain Text part.", "text/plain");
mp.addBodyPart(mp_partPlain);
MimeBodyPart mp_partRich = new MimeBodyPart();
mp_partRich
        .setContent(
                "<html><head></head><body><b><i>This is an HTML
part.</i></b></body></html>",
                "text/html");
mp.addBodyPart(mp_partRich);
message.setContent(mp, "multipart/alternative");
```

61.3.1.3 Creating Delivery Channel-Specific Payloads in a Single Message for Recipients with Different Delivery Types

When sending a message to a destination address, there could be multiple channels involved. Oracle UMS application developers are required to specify the correct multipart format for each channel.

Example 61–4 shows how to create delivery channel (`DeliveryType`) specific payloads in a single message for recipients with different delivery types.

Each top-level part of a multiple payload multipart/alternative message should contain one or more values of this header. The value of this header should be the name of a valid delivery type. Refer to the available values for *DeliveryType* in the enum `DeliveryType`.

Example 61–4 Creating Delivery Channel-specific Payloads in a Single Message for Recipients with Different Delivery Types

```
Message message = MessagingFactory.createMessage();

// create a top-level multipart/alternative MimeMultipart object.
MimeMultipart mp = new MimeMultipart("alternative");

// create first part for SMS payload content.
MimeBodyPart part1 = new MimeBodyPart();
part1.setContent("Text content for SMS.", "text/plain");

part1.setHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "SMS");

// add first part
mp.addBodyPart(part1);

// create second part for EMAIL and IM payload content.
MimeBodyPart part2 = new MimeBodyPart();
MimeMultipart part2_mp = new MimeMultipart("alternative");
MimeBodyPart part2_mp_partPlain = new MimeBodyPart();
part2_mp_partPlain.setContent("Text content for EMAIL/IM.", "text/plain");
part2_mp.addBodyPart(part2_mp_partPlain);
MimeBodyPart part2_mp_partRich = new MimeBodyPart();
part2_mp_partRich.setContent("<html><head></head><body><b><i>" + "HTML content for
EMAIL/IM." +
"</i></b></body></html>", "text/html");
part2_mp.addBodyPart(part2_mp_partRich);
part2.setContent(part2_mp, "multipart/alternative");

part2.addHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "EMAIL");
part2.addHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "IM");

// add second part
mp.addBodyPart(part2);

// set the content of the message
message.setContent(mp, "multipart/alternative");

// set the MultiplePayload flag to true
message.setMultiplePayload(true);
```

61.3.2 API Reference for Class MessagingFactory

The API reference for class `MessagingFactory` can be accessed from the Javadoc.

61.3.3 API Reference for Interface Message

The API reference for interface Message can be accessed from the Javadoc.

61.3.4 API Reference for Enum DeliveryType

The API reference for enum `DeliveryType` can be accessed from the Javadoc.

61.3.5 Addressing a Message

This section describes type of addresses and how to create address objects.

61.3.5.1 Types of Addresses

There are two types of addresses, *device addresses* and *user addresses*. A device address can be of various types, such as email addresses, instant messaging addresses, and telephone numbers. User addresses are user IDs in a user repository.

61.3.5.2 Creating Address Objects

You can address senders and recipients of messages by using the class `MessagingFactory` to create `Address` objects defined by the `Address` interface.

61.3.5.2.1 Creating a Single Address Object Example 61–5 shows code for creating a single `Address` object:

Example 61–5 Creating a Single Address Object

```
Address recipient = MessagingFactory.createAddress("Email:john.doe@oracle.com");
```

61.3.5.2.2 Creating Multiple Address Objects in a Batch Example 61–6 shows code for creating multiple `Address` objects in a batch:

Example 61–6 Creating Multiple Address Objects in a Batch

```
String[] recipientsStr = {"Email:john.doe@oracle.com",
"IM:jabber|john.doe@oracle.com"};
Address[] recipients = MessagingFactory.createAddress(recipientsStr);
```

61.3.5.2.3 Adding Sender or Recipient Addresses to a Message Example 61–7 shows code for adding sender or recipient addresses to a message:

Example 61–7 Adding Sender or Recipient Addresses to a Message

```
Address sender = MessagingFactory.createAddress("Email:john.doe@oracle.com");
Address recipient = MessagingFactory.createAddress("Email:jane.doe@oracle.com");
message.addSender(sender);
message.addRecipient(recipient);
```

61.3.5.3 Creating a Recipient with a Failover Address

Example 61–8 shows code for creating a recipient with a failover address:

Example 61–8 Creating a Single Address Object with Failover

```
String recipientWithFailoverStr = "Email:john.doe@oracle.com,
IM:jabber|john.doe@oracle.com";
Address recipient = MessagingFactory.createAddress(recipientWithFailoverStr);
```

61.3.5.4 API Reference for Class MessagingFactory

The API reference for class `MessagingFactory` can be accessed from the Javadoc.

61.3.5.5 API Reference for Interface Address

The API reference for interface `Address` can be accessed from the Javadoc.

61.3.6 User Preference Based Messaging

When sending a message to a user recipient (to leverage the user's messaging preferences), you can pass facts (current values) for various business terms in the message as metadata. The UMS server matches the supplied facts in the message against conditions for business terms specified in the user's messaging filters.

> **Note:** All facts must be added as metadata in the `Message.NAMESPACE_NOTIFICATION_PREFERENCES` namespace. Metadata in other namespaces are ignored (for resolving user messaging preferences).

Example 61–9 shows how to specify a user recipient and supply facts for business terms for the user preferences in a message. For a complete list of supported business terms, refer to Chapter 64, "User Messaging Preferences."

Example 61–9 User Preference Based Messaging

```
Message message = MessagingFactory.createMessage();
// create and add a user recipient
Address userRecipient1 = MessagingFactory.createAddress("USER:sampleuser1");
message.addRecipient(userRecipient1);
// specify business term facts
message.setMetaData(Message.NAMESPACE_NOTIFICATION_PREFERENCES, "Customer
Name", "ACME");
//  where "Customer Name" is the Business Term name, and "ACME" is the
Business Term value (i.e, fact).
```

61.4 Retrieving Message Status

After sending a message, you can use Oracle UMS to retrieve the message status either synchronously or asynchronously.

61.4.1 Synchronous Retrieval of Message Status

To perform a synchronous retrieval of current status, use the following flow from the `MessagingClient` API:

```
String messageId = messagingClient.send(message);
Status[] statuses = messagingClient.getStatus(messageId);
```

or,

```
Status[] statuses = messagingClient.getStatus(messageId, address[]) --- where
 address[] is an array of one or more of the recipients set in the message.
```

61.4.2 Asynchronous Receiving of Message Status

When asynchronously receiving status, the client application specifies a `Listener` object and an optional correlator object. When incoming status arrives, the listener's `onStatus` callback is invoked. The originally-specified correlator object is also passed to the callback method.

61.4.2.1 Creating a Listener Programmatically

Listeners are purely programmatic. You create a listener by implementing the `oracle.sdp.messaging.Listener` interface. You can implement it as any concrete class - one of your existing classes, a new class, or an anonymous or inner class.

The following code example shows how to implement a status listener:

```
import oracle.sdp.messaging.Listener;

    public class StatusListener implements Listener {

      @Override
      public void onMessage(Message message, Serializable correlator) {
      }
      @Override
      public void onStatus(Status status, Serializable correlator) {
          System.out.println("Received Status: " + status + " with optional
correlator: " +

correlator);
      }
    }
```

You pass a reference to the `Listener` object to the `setStatusListener` or `send` methods, as described in "Default Status Listener" and "Per Message Status Listener". When a status arrives for your message, the UMS infrastructure invokes the Listener's `onStatus` method as appropriate.

61.4.2.2 Default Status Listener

The client application typically sets a default status listener (Example 61–10). When the client application sends a message, delivery status callbacks for the message invoke the default listener's `onStatus` method.

Example 61–10 Default Status Listener

```
messagingClient.setStatusListener(new MyStatusListener());
messagingClient.send(message);
```

61.4.2.3 Per Message Status Listener

In this approach, the client application sends a message and specifies a Listener object and an optional correlator object (Example 61–11). When delivery status callbacks are available for that message, the specified listener's `onStatus` method is invoked. The originally-specified correlator object is also passed to the callback method.

Example 61–11 Per Message Status Listener

```
messagingClient.send(message, new MyStatusListener(), null);
```

61.5 Receiving a Message

This section describes how an application receives messages. To receive a message you must first register an access point. From the application perspective there are two modes for receiving a message, synchronous and asynchronous.

61.5.1 Registering an Access Point

The client application can create and register an access point, specifying that it wants to receive incoming messages sent to a particular address. Since the client application has not specified any message listeners, any received messages are held by UMS. The client application can then invoke the receive method to fetch the pending messages. When receiving messages without specifying an access point, the application receives messages for any of the access points that it has registered. Otherwise, if an access point is specified, the application receives messages sent to that access point.

`AccessPoint` represents one or more device addresses to receive incoming messages. An application that wants to receive incoming messages must register one or more access points that represent the recipient addresses of the messages. The server matches the recipient address of an incoming message against the set of registered access points, and routes the incoming message to the application that registered the matching access point.

You can use `MessagingFactory.createAccessPoint` to create an access point and `MessagingClient.registerAccessPoint` to register it for receiving messages.

To register an SMS access point for the number `9000`:

```
AccessPoint accessPointSingleAddress =
 MessagingFactory.createAccessPoint(AccessPoint.AccessPointType.SINGLE_ADDRESS,
 DeliveryType.SMS, "9000");
messagingClient.registerAccessPoint(accessPointSingleAddress);
```

To register SMS access points in the number range `9000` to `9999`:

```
AccessPoint accessPointRangeAddress =
 MessagingFactory.createAccessPoint(AccessPoint.AccessPointType.NUMBER_RANGE,
 DeliveryType.SMS,"9000,9999");
messagingClient.registerAccessPoint(accessPointRangeAddress);
```

61.5.2 Synchronous Receiving

A receive is a nonblocking operation. If there are no pending messages for the application or access point, the call returns immediately with an empty list. Receive is not guaranteed to return all available messages, but may return only a subset of available messages for efficiency reasons.

You can use the method `MessagingClient.receive` to synchronously receive messages. This is a convenient polling method for light-weight clients that do not want the configuration overhead associated with receiving messages asynchronously. This method returns a list of messages that are immediately available in the application inbound queue.

It performs a nonblocking call, so if no message is currently available, the method returns null.

Note: A single invocation does not guarantee retrieval of all available messages. You must poll to ensure receiving all available messages.

61.5.3 Asynchronous Receiving

When asynchronously receiving messages, the client application registers an access point and specifies a `Listener` object and an optional correlator object. When incoming messages arrive at the specified access point address, the listener's `onMessage` callback is invoked. The originally-specified correlator object is also passed to the callback method.

61.5.3.1 Creating a Listener Programmatically

Listeners are purely programmatic. You create a listener by implementing the `oracle.sdp.messaging.Listener` interface. You can implement it as any concrete class - one of your existing classes, a new class, or an anonymous or inner class.

The following code example shows how to implement a message listener:

```
import oracle.sdp.messaging.Listener;

    public class MyListener implements Listener {

      @Override
      public void onMessage(Message message, Serializable correlator) {
          System.out.println("Received Message: " + message + " with optional
correlator: " +
correlator);
      }
      @Override
      public void onStatus(Status status, Serializable correlator) {
          System.out.println("Received Status: " + status + " with optional
correlator: " +
correlator);
      }

    }
```

You pass a reference to the Listener object to the `setMessageListener` or `registerAccessPoint` methods, as described in "Default Message Listener" and "Per Access Point Message Listener". When a message arrives for your application, the UMS infrastructure invokes the Listener's `onMessage` method.

61.5.3.2 Default Message Listener

The client application typically sets a default message listener (Example 61–12). This listener is invoked for any delivery statuses for messages sent by this client application that do not have an associated listener. When Oracle UMS receives messages addressed to any access points registered by this client application, it invokes the `onMessage` callback for the client application's default listener.

To remove a default listener, call this method with a null argument.

Example 61–12 Default Message Listener

```
messagingClient.setMessageListener(new MyListener());
```

See the sample application `usermessagingsample-echo` for detailed instructions on asynchronous receiving.

61.5.3.3 Per Access Point Message Listener

The client application can also register an access point and specify a `Listener` object and an optional correlator object (Example 61–13). When incoming messages arrive at the specified access point address, the specified listener's `onMessage` method is invoked. The originally-specified correlator object is also passed to the callback method.

Example 61–13 Per Access Point Message Listener

```
messagingClient.registerAccessPoint(accessPoint, new MyListener(), null);
```

61.5.4 Message Filtering

A `MessageFilter` is used by an application to exercise greater control over what messages are delivered to it. A `MessageFilter` contains a matching criterion and an action. An application can register a series of message filters; they are applied in order against an incoming (received) message; if the criterion matches the message, the action is taken. For example, an application can use `MessageFilters` to implement necessary blacklists, by rejecting all messages from a given sender address.

You can use `MessagingFactory.createMessageFilter` to create a message filter, and `MessagingClient.registerMessageFilter` to register it. The filter is added to the end of the current filter chain for the application. When a message is received, it is passed through the filter chain in order; if the message matches a filter's criterion, the filter's action is taken immediately. If no filters match the message, the default action is to accept the message and deliver it to the application.

For example, to reject a message with the subject `"spam"`:

```
MessageFilter subjectFilter = MessagingFactory.createMessageFilter("spam",
 MessageFilter.FieldType.SUBJECT, null, MessageFilter.Action.REJECT);
messagingClient.registerMessageFilter(subjectFilter);
```

To reject messages from email address `spammer@foo.com`:

```
MessageFilter senderFilter =
 MessagingFactory.createBlacklistFilter("spammer@foo.com");
messagingClient.registerMessageFilter(senderFilter);
```

61.6 Configuring for a Cluster Environment

The API supports an environment where client applications and the UMS server are deployed in a cluster environment. For a clustered deployment to function as expected, client applications must be configured correctly. The following rules apply:

- Two client applications are considered to be instances of the same application if they use the same `ApplicationName` configuration parameter. Typically this parameter is synthesized by the API implementation and does not need to be populated by the application developer.
- Instances of the same application share most of their configuration, and artifacts such as `Access` Points and Message Filters that are registered by one instance are shared by all instances.
- The `ApplicationInstanceName` configuration parameter enables you to distinguish instances from one another. Typically this parameter is synthesized by

the API implementation and does not need to be populated by the application developer. Refer to the Javadoc for cases in which this value must be populated.

- Application sessions are instance-specific. You can set the session flag on a message to ensure that any reply is received by the instance that sent the message.
- Listener correlators are instance-specific. If two different instances of an application register listeners and supply different correlators, then when instance A' s listener is invoked, correlator A is supplied; when instance B' s listener is invoked, correlator B is supplied.

61.7 Configuring Security

Client applications may need to specify one or more additional configuration parameters (described in Table 61–1) to establish a secure listener.

61.8 Threading Model

Client applications that use the UMS Java API are usually multithreaded. Typical scenarios include a pool of EJB instances, each of which uses a `MessagingClient` instance; and a servlet instance that is serviced by multiple threads in a web container. The UMS Java API supports the following thread model:

- Each call to `MessagingClientFactory.createMessagingClient` returns a new `MessagingClient` instance.
- When two `MessagingClient` instances are created by passing parameter maps that are equal to `MessagingClientFactory.createMessagingClient`, they are instances of the same client. Instances created by passing different parameter maps are instances of separate clients.
- An instance of `MessagingClient` is not thread safe when it has been obtained using `MessagingClientFactory.createMessagingClient`. Client applications must ensure that a given instance is used by only one thread at a time. They may do so by ensuring that an instance is only visible to one thread at a time, or by synchronizing access to the `MessagingClient` instance.
- Two instances of the same client (created with identical parameter maps) do share some resources – notably they share Message and Status Listeners, and use a common pool of Worker threads to execute asynchronous messaging operations. For example, if instance A calls `setMessageListener()`, and then instance B calls `setMessageListener()`, then B's listener is the active default message listener.

The following are typical use cases:

- To use the UMS Java API from an EJB (either a Message Driven Bean or a Session Bean) application, the recommended approach is to create a `MessagingClient` instance in the bean' s `ejbCreate` (or equivalent `@PostConstruct`) method, and store the `MessagingClient` in an instance variable in the bean class. The EJB container ensures that only one thread at a time uses a given EJB instance, which ensures that only one thread at a time accesses the bean' s `MessagingClient` instance.
- To use the UMS Java API from a Servlet, there are several possible approaches. In general Web containers create a single instance of the servlet class, which may be accessed by multiple threads concurrently. If a single `MessagingClient` instance is created and stored in a servlet instance variable, then access to the instance must be synchronized.

Another approach is to create a pool of `MessagingClient` instances that are shared among servlet threads.

Finally, you can associate individual `MessagingClient` instances with individual HTTP Sessions. This approach allows increased concurrency compared to having a single `MessagingClient` for all servlet requests. However, it is possible for multiple threads to access an HTTP Session at the same time due to concurrent client requests, so synchronization is still required in this case.

61.8.1 Listener Threading

You can achieve asynchronous listening by spawning one or more worker threads that listen to the configured JMS queues for incoming messages and statuses. By default, one worker thread is spawned for incoming messages, and one worker thread is spawned for incoming status notifications (assuming at least one message or status listener is registered, respectively). Client applications can increase the concurrency of asynchronous processing by configuring additional worker threads. This is done by specifying integer values for the `MessagingConstants.MESSAGE_LISTENER_THREADS` and `MessagingConstants.STATUS_LISTENER_THREADS` keys, settings these values to the desired number of worker threads in the configuration parameters used when creating a `MessagingClient` instance.

61.9 Using the UMS Client API to Build a Client Application

This section describes how to create an application called *usermessagingsample*, a web client application that uses the UMS Client API for both outbound messaging and the synchronous retrieval of message status. *usermessagingsample* also supports inbound messaging. Once you have deployed and configured *usermessagingsample*, you can use it to send a message to an email client.

Note: To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL:
`https://soasamples.samplecode.oracle.com/`

Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

This sample focuses on a Web Application Module (WAR), which defines some HTML forms and servlets. You can examine the code and corresponding XML files for the web application module from the provided `usermessagingsample-src.zip` source. The servlets uses the UMS Client API to create an UMS Client instance (which in turn registers the application's information) and sends messages.

This application, which is packaged as a Enterprise ARchive file (EAR) called *usermessagingsample.ear*, has the following structure:

- `usermessagingsample.ear`
  - `META-INF`
    - `application.xml` -- Descriptor file for all of the application modules.
    - `weblogic-application.xml` -- Descriptor file that contains the `import` of the `oracle.sdp.messaging` shared library.

- `usermessagingsample-web.ear` -- Contains the web-based front-end and servlets.
  - `WEB-INF`
    - `web.xml`
    - `weblogic.xml`

The prebuilt sample application, and the source code (usermessagingsample-src.zip) are available on OTN.

61.9.1 Overview of Development

The following steps describe the process of building an application capable of outbound messaging using `usermessagingsample.ear` as an example:

1. Section 61.9.2, "Configuring the Email Driver"
2. Section 61.9.3, "Using JDeveloper 11g to Build the Application"
3. Section 61.9.4, "Deploying the Application"
4. Section 61.9.5, "Testing the Application"

61.9.2 Configuring the Email Driver

To enable the Oracle User Messaging Service's email driver to perform outbound messaging and status retrieval, configure the email driver as follows:

- Enter the name of the SMTP mail server as the value for the `OutgoingMailServer` property.

> **Note:** This sample application is generic and can support outbound messaging through other channels when the appropriate messaging drivers are deployed and configured.

61.9.3 Using JDeveloper 11g to Build the Application

This section describes using a Windows-based build of JDeveloper to build, compile, and deploy *usermessagingsample* through the following steps:

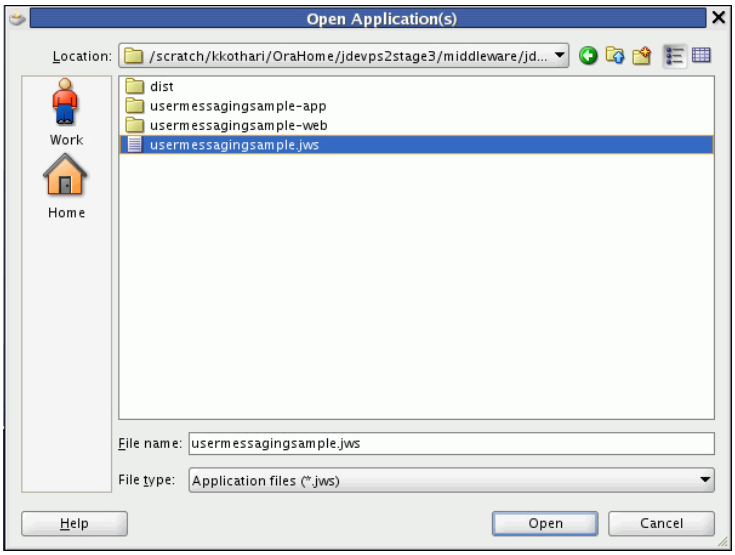
61.9.3.1 Opening the Project

1. Unzip `usermessagingsample-src.zip`, to the *`JDEV_HOME`*`/communications/samples/` directory. This directory must be used for the shared library references to be valid in the project.

> **Note:** If you choose to use a different directory, you must update the `oracle.sdp.messaging` library source path to *`JDEV_HOME`*`/communications/modules/oracle.sdp.messaging_11.1.1/sdpmessaging.jar`.

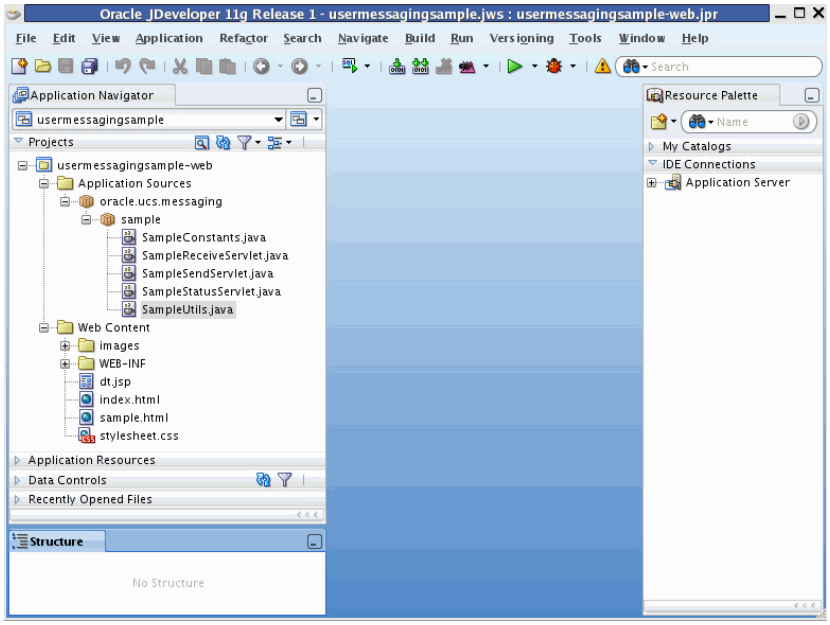
2. Open `usermessagingsample.jws` (contained in the .zip file) in Oracle JDeveloper.

Figure 61–1 Oracle JDeveloper Open Application Window



In the Oracle JDeveloper main window, the project appears.

Figure 61–2 Oracle JDeveloper Main Window

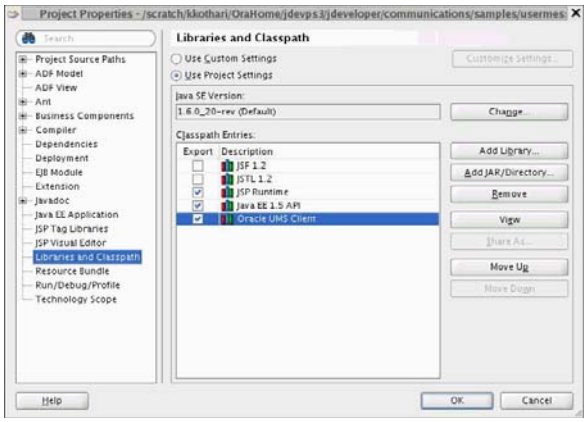


3. Satisfy the build dependencies for the sample application by ensuring the "Oracle UMS Client" library is used by the Web module.

    1. In the Application Navigator, right-click web module **usermessagingsample-web**, and select **Project Properties**.

    2. In the left pane, select **Libraries and Classpath**.

Figure 61–3 Verifying Libraries



3. Click **OK**.

4. Explore the Java files under the **usermessagingsample-web** project to see how the messaging client APIs are used to send messages, get statuses, and synchronously receive messages. The MessagingClient instance is created in `SampleUtils.java` in the project.

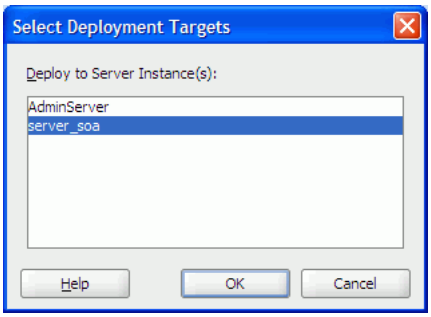
61.9.4 Deploying the Application

Perform the following steps to deploy the application:

1. Create an Application Server Connection by right-clicking the application in the navigation pane and selecting New. Follow the instructions in Section 61.11, "Creating a New Application Server Connection."

2. Deploy the application by selecting the **usermessagingsample application**, **Deploy**, **usermessagingsample**, **to**, and **SOA_server** (Figure 61–4).

Figure 61–4 Deploying the Project



3. Verify that the message `Build Successful` appears in the log.

4. Verify that the message `Deployment Finished` appears in the deployment log.

   You have successfully deployed the application.

   Before you can run the sample, you must configure any additional drivers in Oracle User Messaging Service and optionally configure a default device for the user receiving the message in User Messaging Preferences.

Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

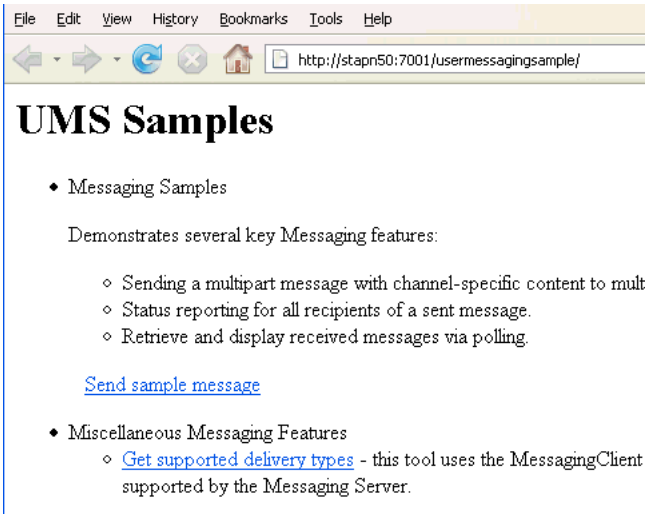
61.9.5 Testing the Application

Once **usermessagingsample** has been deployed to a running instance of Oracle WebLogic Server, perform the following:

1. Launch a web browser and enter the address of the sample application as follows: `http://host:http-port/usermessagingsample/`. For example, enter `http://localhost:7001/usermessagingsample/` into the browser's navigation bar.

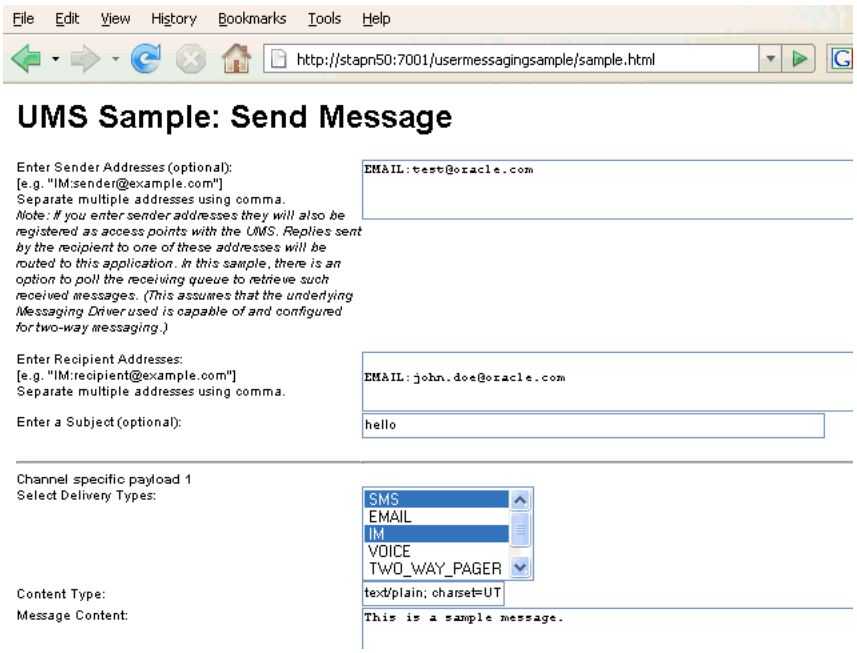
   When prompted, enter login credentials. For example, username `weblogic`. The browser page for testing messaging samples appears (Figure 61–5).

Figure 61–5 Testing the Sample Application



2. Click **Send sample message**. The Send Message page appears (Figure 61–6).

Figure 61–6 Addressing the Test Message



3. As an optional step, enter the sender address in the following format:

   `Email:`*`sender_address`*.

   For example, enter `Email:sender@oracle.com`.

4. Enter one or more recipient addresses. For example, enter `Email:recipient@oracle.com`. Enter multiple addresses as a comma-separated list as follows:

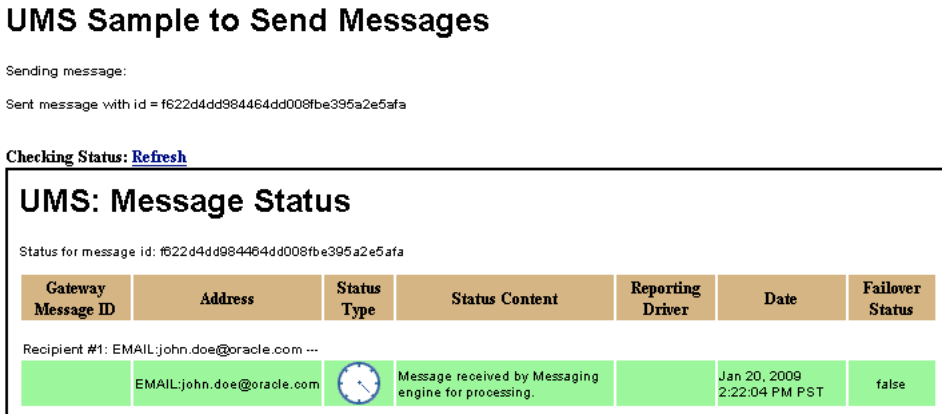
   `Email:`*`recipient_address1`*`, Email:`*`recipient_address2`*.

   If you have configured user messaging preferences, you can address the message simply to `User:`*`username`*. For example, `User:weblogic`.

5. As an optional step, enter a subject line or content for the email.

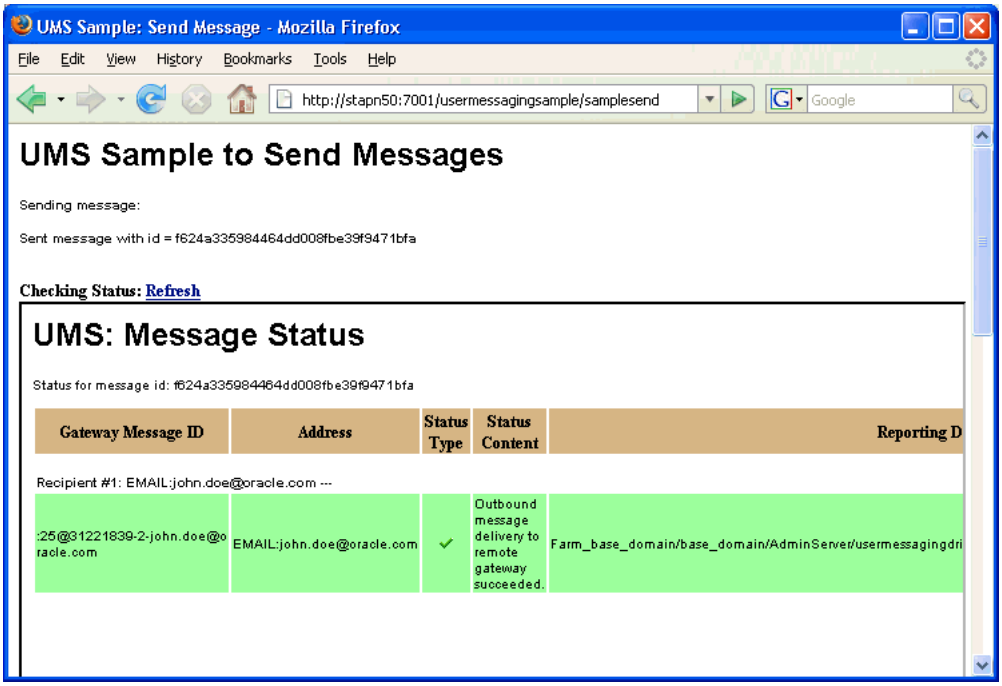
6. Click **Send**. The Message Status page appears, showing the progress of transaction (Message received by Messaging engine for processing in Figure 61–7).

Figure 61–7 Message Status



7. Click **Refresh** to update the status. When the email message has been delivered to the email server, the *Status Content* field displays *Outbound message delivery to remote gateway succeeded.*, as illustrated in Figure 61–8.

Figure 61–8 Checking the Message Status



61.10 Using the UMS Client API to Build a Client Echo Application

This section describes how to create an application called usermessagingsample-echo, a demo client application that uses the UMS Client API to asynchronously receive messages from an email address and echo a reply back to the sender.

> **Note:** To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL:
> `https://soasamples.samplecode.oracle.com/`
>
> Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

This application, which is packaged as a Enterprise Archive file (EAR) called *usermessagingsample-echo.ear*, has the following structure:

- `usermessagingsample-echo.ear`
  - `META-INF`
    - `application.xml` -- Descriptor file for all of the application modules.
    - `weblogic-application.xml` -- Descriptor file that contains the `import` of the `oracle.sdp.messaging` shared library.

- `usermessagingsample-echo-web.war` -- Contains the web-based front-end and servlets. It also contains the listener that processes a received message and returns an echo response
  - `WEB-INF`
    - `web.xml`
    - `weblogic.xml`

The prebuilt sample application, and the source code (`usermessagingsample-echo-src.zip`) are available on OTN.

61.10.1 Overview of Development

The following steps describe the process of building an application capable of asynchronous inbound and outbound messaging using `usermessagingsample-echo.ear` as an example:

1. Section 61.10.2, "Configuring the Email Driver"
2. Section 61.10.3, "Using JDeveloper 11g to Build the Application"
3. Section 61.10.4, "Deploying the Application"
4. Section 61.10.5, "Testing the Application"

61.10.2 Configuring the Email Driver

To enable the Oracle User Messaging Service's email driver to perform inbound and outbound messaging and status retrieval, configure the email driver as follows:

- Enter the name of the SMTP mail server as the value for the **OutgoingMailServer** property.
- Enter the name of the IMAP4/POP3 mail server as the value for the **IncomingMailServer** property. Also, configure the incoming user name, and password.

> **Note:** This sample application is generic and can support inbound and outbound messaging through other channels when the appropriate messaging drivers are deployed and configured.

61.10.3 Using JDeveloper 11g to Build the Application

This section describes using a Windows-based build of JDeveloper to build, compile, and deploy **usermessagingsample-echo** through the following steps:

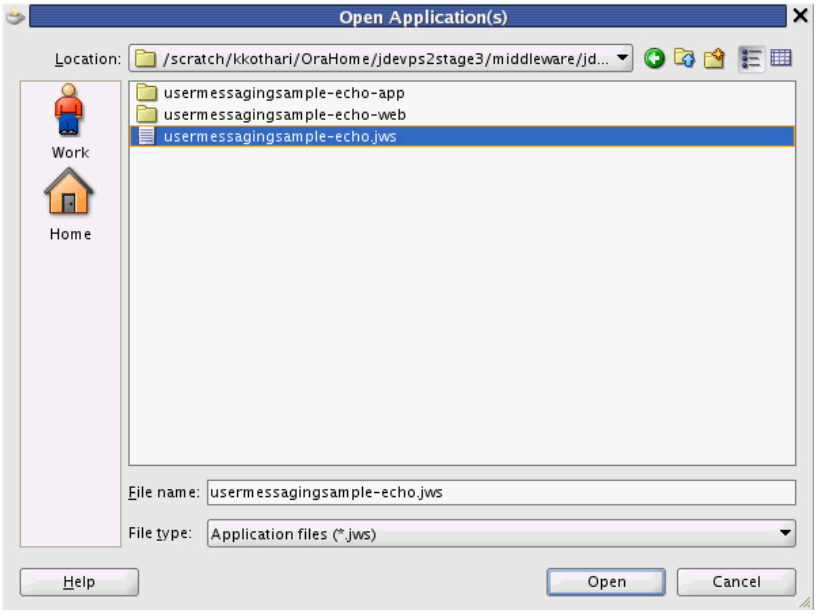
61.10.3.1 Opening the Project

1. Unzip `usermessagingsample-echo-src.zip`, to the *`JDEV_HOME`*`/communications/samples/` directory. This directory must be used for the shared library references to be valid in the project.

> **Note:** If you choose to use a different directory, you must update the oracle.sdp.messaging library source path to *`JDEV_HOME`*`/communications/modules/oracle.sdp.messaging_11.1.1/sdpmessaging.jar`.

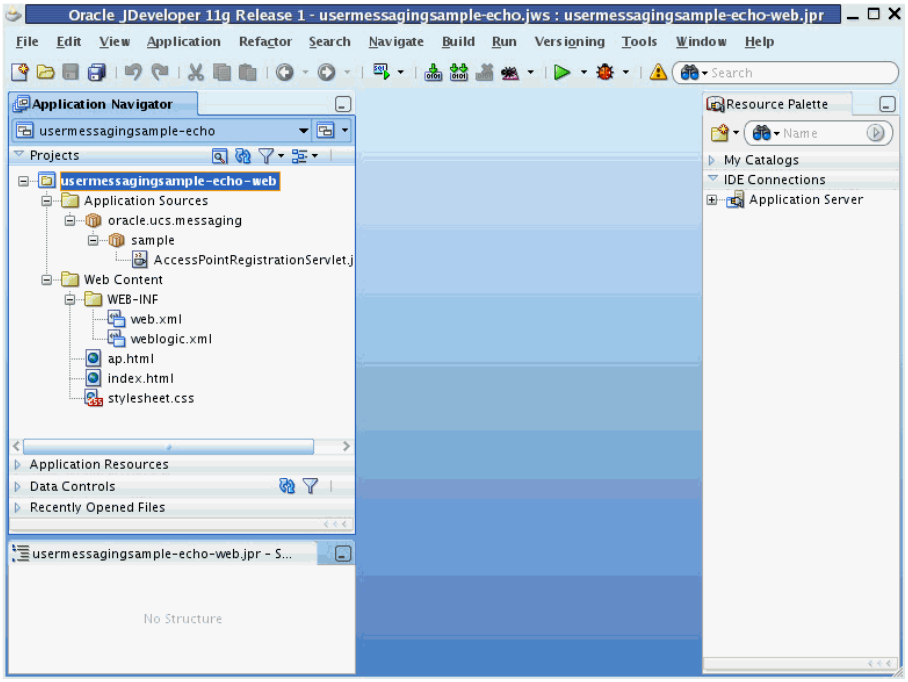
2. Open `usermessagingsample-echo.jws` (contained in the .zip file) in Oracle JDeveloper (Figure 61–9).

Figure 61–9 Opening the Project



In the Oracle JDeveloper main window the project appears (Figure 61–10).

Figure 61–10 Oracle JDeveloper Main Window



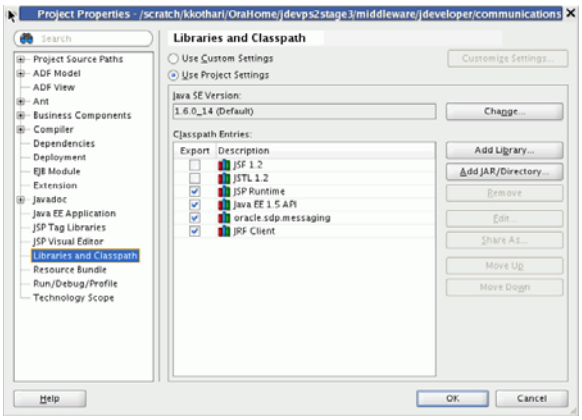
3. Verify that the build dependencies for the sample application have been satisfied by checking that the following library has been added to the `usermessagingsample-echo-web` module.

- Library: *oracle.sdp.messaging*, Classpath: `JDEV_HOME/communications/modules/oracle.sdp.messaging_11.1.1/sdpmessaging.jar`. This is the Java library used by UMS and applications that use UMS to send and receive messages.

Perform the following steps for each module:

1. In the Application Navigator, right-click the module and select **Project Properties**.
2. In the left pane, select **Libraries and Classpath** (Figure 61–11).

Figure 61–11 Verifying Libraries



3. Click **OK**.

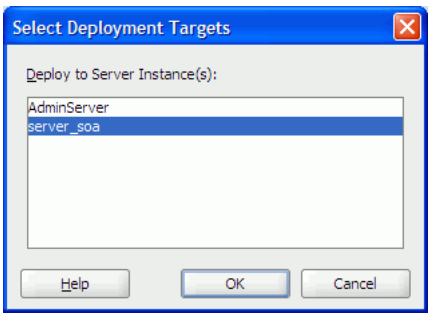
4. Explore the Java files under the **usermessagingsample-echo-web** project to see how the messaging client APIs are used to register and unregister access points, and how the `EchoListener` is used to asynchronously receive messages.

61.10.4 Deploying the Application

Perform the following steps to deploy the application:

1. Create an Application Server Connection by right-clicking the application in the navigation pane and selecting New. Follow the instructions in Section 61.11, "Creating a New Application Server Connection."
2. Deploy the application by selecting the **usermessagingsample-echo application**, **Deploy**, **usermessagingsample-echo**, **to**, and **SOA_server** (Figure 61–12).

Figure 61–12 Deploying the Project



3. Verify that the message `Build Successful` appears in the log.

4. Verify that the message `Deployment Finished` appears in the deployment log.

   You have successfully deployed the application.

   Before you can run the sample you must configure any additional drivers in Oracle User Messaging Service and optionally configure a default device for the user receiving the message in User Messaging Preferences.

   Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

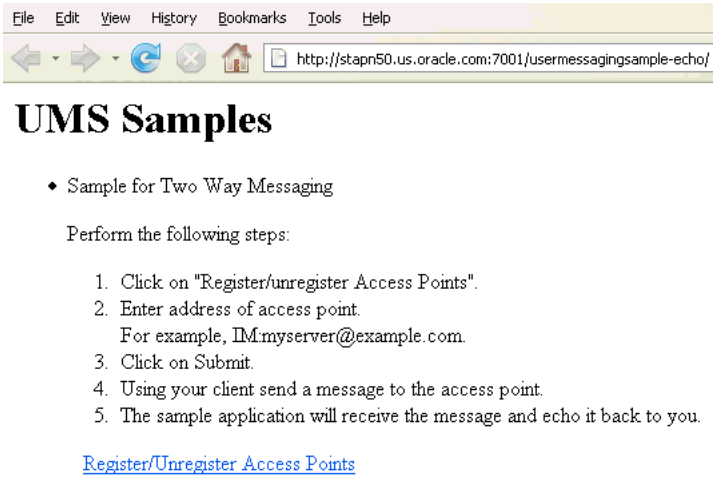
61.10.5 Testing the Application

Once **usermessagingsample-echo** has been deployed to a running instance of Oracle WebLogic Server, perform the following:

1. Launch a web browser and enter the address of the sample application as follows: `http://host:http-port/usermessagingsample-echo/`. For example, enter `http://localhost:7001/usermessagingsample-echo/` into the browser's navigation bar.

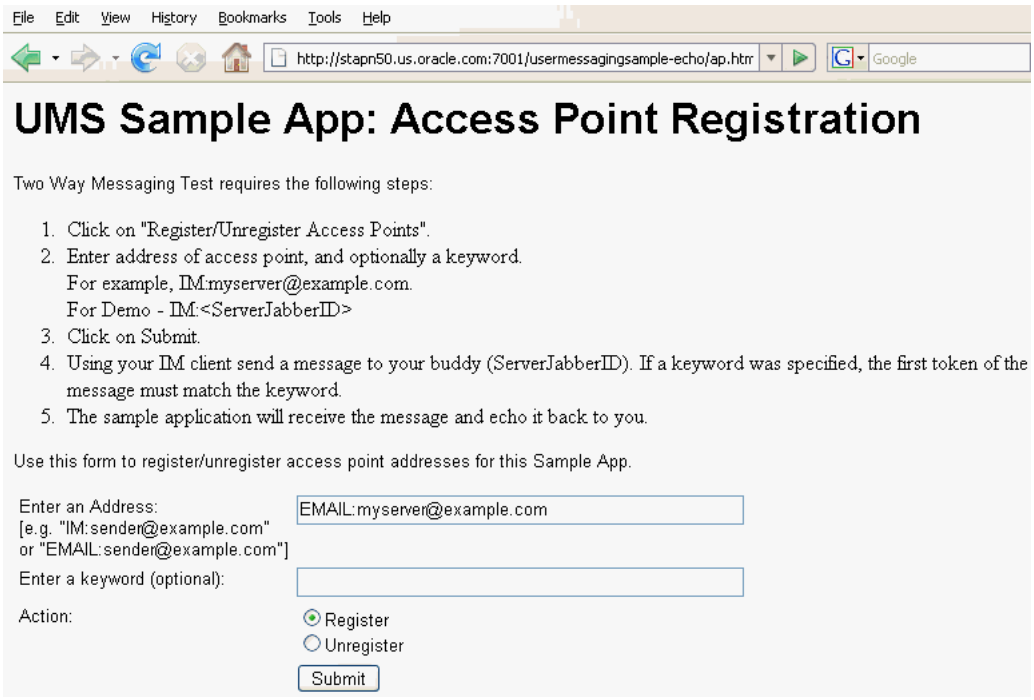
   When prompted, enter login credentials. For example, username `weblogic`. The browser page for testing messaging samples appears (Figure 61–13).

Figure 61–13 Testing the Sample Application



2. Click **Register/Unregister Access Points**. The *Access Point Registration* page appears (Figure 61–14).

Figure 61–14 Registering an Access Point



3. Enter the access point address in the following format:

   `EMAIL:`*`server_address`*.

   For example, enter `EMAIL:myserver@example.com`.

4. Select the Action **Register** and Click **Submit**. The registration status page appears, showing "Registered" in Figure 61–15).

Figure 61–15 Access Point Registration Status

5. Send a message from your messaging client (for email, your email client) to the address you just registered as an access point in the previous step.

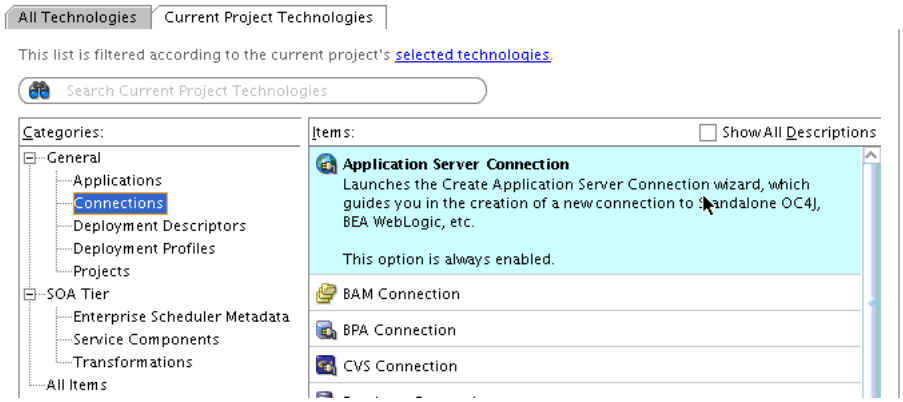
   If the UMS messaging driver for that channel is configured correctly, you should expect to receive an echo message back from the **usermessagingsample-echo** application.

61.11 Creating a New Application Server Connection

Perform the following steps to create an Application Server Connection.

1. Create a new Application Server Connection by right-clicking the project and selecting **New**, **Connections**, and **Application Server Connection** (Figure 61–16).

Figure 61–16 New Application Server Connection



2. Name the connection `SOA_server` and click **Next** (Figure 61–17).
3. Select **WebLogic 10.3** as the **Connection Type**.

Figure 61–17 New Application Server Connection

4. Enter the authentication information. A typical value for user name is `weblogic`.
5. In the Connection dialog, enter the hostname, port, and SSL port for the SOA admin server, and enter the name of the domain for WLS Domain.
6. Click **Next**.
7. In the Test dialog, click **Test Connection**.
8. Verify that the message `Success!` appears.

   The Application Server Connection has been created.

62

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

This chapter describes how to use the User Messaging Service (UMS) Web Service API to develop applications. This API serves as a programmatic entry point for Fusion Middleware application developers to implement UMS messaging applications that run in a remote container relative to the UMS server.

This chapter includes the following sections:

Note: To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL:
`https://soasamples.samplecode.oracle.com/`

Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

62.1 Introduction to the UMS Web Service API

The UMS Web Service API is functionally identical to the Java API. The JAX-WS and JAXB bindings of the web service types and interfaces are named similarly to the corresponding Java API classes, but are in their own package space. Classes from the two APIs are not interoperable.

Consumers of the API can get an instance of a `MessagingClient` object using a factory method. The deployment is as a shared library, "oracle.sdp.client".

The UMS Web Service API consists of packages grouped as follows:

- Common and Client Packages
  - `oracle.ucs.messaging.ws`
  - `oracle.ucs.messaging.ws.types`
- Web Service API Web Service Definition Language (WSDL) files:
  - messaging.wsdl:defines the operations invoked by a Web service client.
  - listener.wsdl:defines the callback operations that a client must implement to receive asynchronous message or status notifications.

The samples with source code are available on Oracle Technology Network (OTN).

62.2 Creating a UMS Client Instance and Specifying Runtime Parameters

This section describes the requirements for creating a UMS Client. You can create a instance of `oracle.ucs.messaging.ws.MessagingClient` by using the public constructor. Client applications can specify a set of parameters at runtime when instantiating a client object. For example, you configure a `MessagingClient` instance by specifying parameters as a map of key-value pairs in a `java.util.Map<String, Object>`. Among other things, the configuration parameters serve to identify the Web Service endpoint URL identifying the UMS server to communicate with, and other Web Service-related information such as security policies. Client applications are responsible for storing and loading the configuration parameters using any available mechanism.

You are responsible for mapping the parameters to/from whatever configuration storage mechanism is appropriate for your deployment. The `MessagingClient` class uses the specified key/value pairs for configuration, and passes through all parameters to the underlying JAX-WS service. Any parameters recognized by JAX-WS are valid. Table 62–1 lists the most common configuration parameters:

Table 62–1 Configuration Parameters Specified at Runtime

Key	Use
`javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTY`	Endpoint URL for the remote UMS WS. This is typically "http://<host>:<port>/ucs/messaging/webservice".
`javax.xml.ws.BindingProvider.USERNAME_PROPERTY`	Username to be asserted in WS-Security headers when relevant
`oracle.ucs.messaging.ws.ClientConstants.POLICIES`	Set of OWSM WS-Security policies to attach to the client's requests. These must match the policies specified on the server side.
oracle.wsm.security.util.SecurityConstants.Config.KEYSTORE_RECIPIENT_ALIAS_PROPERTY	Used for OWSM policy attachment. Specifies an alternate alias to use for looking up encryption and signing keys from the credential store.
oracle.wsm.security.util.SecurityConstants.ClientConstants.WSS_CSF_KEY	Used for OWSM policy attachment. Specifies a credential store key to use for looking up remote username/password information from the Oracle Web Services Management credential store map.

A `MessagingClient` cannot be reconfigured after it is instantiated. Instead, a new instance of the `MessagingClient` class must be created using the new configuration.

Example 62–1 shows code for creating a `MessagingClient` instance using username/token security, using the programmatic approach:

Example 62–1 Programmatic Approach to Creating a MessagingClient Instance, Username/Token Security

```
HashMap<String, Object> config = new HashMap<String, Object>();
config.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
 "http://example.com:8001/ucs/messaging/webservice");
config.put(ClientConstants.POLICIES, new String[] {"oracle/wss11_username_token_
with_message_protection_client_policy"});
config.put(BindingProvider.USERNAME_PROPERTY, "user1");
config.put(oracle.wsm.security.util.SecurityConstants.Config.CLIENT_CREDS_
LOCATION, oracle.wsm.security.util.SecurityConstants.Config.CLIENT_CREDS_LOC_
SUBJECT);
config.put(oracle.wsm.security.util.SecurityConstants.ClientConstants.WSS_CSF_KEY,
 "user1-passkey");
config.put(MessagingConstants.APPLICATION_NAME, "MyUMSWSApp");
mClient = new MessagingClient(config);
```

Example 62–2 shows code for creating a `MessagingClient` instance using SAML token security, using the programmatic approach:

Example 62–2 Programmatic Approach to Creating a MessagingClient Instance, SAML Token Security

```
HashMap<String, Object> config = new HashMap<String, Object>();
config.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
 "http://example.com:8001/ucs/messaging/webservice");
config.put(ClientConstants.POLICIES, new String[] {"oracle/wss11_saml_token_
identity_switch_with_message_protection_client_policy"});
config.put(BindingProvider.USERNAME_PROPERTY, "user1");
config.put(oracle.wsm.security.util.SecurityConstants.Config.CLIENT_CREDS_
LOCATION, oracle.wsm.security.util.SecurityConstants.Config.CLIENT_CREDS_LOC_
SUBJECT);
config.put(oracle.wsm.security.util.SecurityConstants.Config.KEYSTORE_RECIPIENT_
ALIAS_PROPERTY, "example.com");
config.put(MessagingConstants.APPLICATION_NAME, "MyUMSWSApp");
mClient = new MessagingClient(config);
```

A `MessagingClient` cannot be reconfigured after it is instantiated. Instead, you must create a new instance of the `MessagingClient` class using the desired configuration.

Factory methods are provided for creating Web Service API types in the class "`oracle.ucs.messaging.ws.MessagingFactory`".

62.3 Sending a Message

Invoking the `send` method causes the message to be delivered to UMS and processed accordingly. The `send` method returns a `String message` identifier that the client application can later use to retrieve message delivery status, or to correlate with asynchronous status notifications that are delivered to a Listener. The status returned is the latest known status based on UMS internal processing and delivery notifications received from external gateways.

The types of messages that can be created include plaintext messages, multipart messages that can consist of text/plain and text/html parts, and messages that include

the creation of delivery channel (`DeliveryType`) specific payloads in a single message for recipients with different delivery types.

62.3.1 Creating a Message

This section describes the various types of messages that can be created.

62.3.1.1 Creating a Plaintext Message

Example 62–3 shows how to create a plaintext message using the UMS Web Service API.

Example 62–3 Creating a Plaintext Message Using the UMS Web Service API

```
Message message = MessagingFactory.createTextMessage("This is a Plain Text
 message.");
Message message = MessagingFactory.createMessage();
message.setContent(new DataHandler(new StringDataSource("This is a Plain Text
 message.", "text/plain; charset=UTF-8")));
```

62.3.1.2 Creating a Multipart/Mixed Message (with Text and Binary Parts)

Example 62–4 shows how to create a multipart/mixed message using the UMS Web Service API.

Example 62–4 Creating a Multipart/Mixed Message Using the UMS Web Service API

```
Message message = MessagingFactory.createMessage();
MimeMultipart mp = new MimeMultipart("mixed");

// Create the first body part
MimeBodyPart mp_partPlain = new MimeBodyPart();
StringDataSource plainDS = new StringDataSource("This is a Plain Text part.",
 "text/plain; charset=UTF-8");
mp_partPlain.setDataHandler(new DataHandler(plainDS));
mp.addBodyPart(mp_partPlain);

byte[] imageData;
// Create or load image data in the above byte array (code not shown for brevity)

// Create the second body part
MimeBodyPart mp_partBinary = new MimeBodyPart();
ByteArrayDataSource binaryDS = new ByteArrayDataSource(imageData, "image/gif");
mp_partBinary.setDataHandler(binaryDS);
mp.addBodyPart(mp_partBinary);

message.setContent(new DataHandler(mp, mp.getContentType()));
```

62.3.1.3 Creating a Multipart/Alternative Message (with Text/Plain and Text/HTML Parts)

Example 62–5 shows how to create a multipart/alternative message using the UMS Web Service API.

Example 62–5 Creating a Multipart/Alternative Message Using the UMS Web Service API

```
Message message = MessagingFactory.createMessage();
MimeMultipart mp = new MimeMultipart("alternative");
MimeBodyPart mp_partPlain = new MimeBodyPart();
StringDataSource plainDS = new StringDataSource("This is a Plain Text part.",
```

```
"text/plain; charset=UTF-8");
mp_partPlain.setDataHandler(new DataHandler(plainDS));
mp.addBodyPart(mp_partPlain);

MimeBodyPart mp_partRich = new MimeBodyPart();
StringDataSource richDS = new StringDataSource(
    "<html><head></head><body><b><i>This is an HTML part.</i></b></body></html>",
    "text/html");
mp_partRich.setDataHandler(new DataHandler(richDS));
mp.addBodyPart(mp_partRich);

message.setContent(new DataHandler(mp, mp.getContentType()));
```

62.3.1.4 Creating Delivery Channel-Specific Payloads in a Single Message for Recipients with Different Delivery Types

When sending a message to a destination address, there could be multiple channels involved. Oracle UMS application developers are required to specify the correct multipart format for each channel.

Example 62–6 shows how to create delivery channel (`DeliveryType`) specific payloads in a single message for recipients with different delivery types.

Each top-level part of a multiple payload multipart/alternative message should contain one or more values of this header. The value of this header should be the name of a valid delivery type. Refer to the available values for *DeliveryType* in the enum `DeliveryType`.

Example 62–6 Creating Delivery Channel-specific Payloads in a Single Message for Recipients with Different Delivery Types

```
Message message = MessagingFactory.createMessage();

// create a top-level multipart/alternative MimeMultipart object.
MimeMultipart mp = new MimeMultipart("alternative");

// create first part for SMS payload content.
MimeBodyPart part1 = new MimeBodyPart();
part1.setDataHandler(new DataHandler(new StringDataSource("Text content for SMS.",
 "text/plain; charset=UTF-8")));
part1.setHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "SMS");
// add first part
mp.addBodyPart(part1);

// create second part for EMAIL and IM payload content.
MimeBodyPart part2 = new MimeBodyPart();
MimeMultipart part2_mp = new MimeMultipart("alternative");
MimeBodyPart part2_mp_partPlain = new MimeBodyPart();
part2_mp_partPlain.setDataHandler(new DataHandler(new StringDataSource("Text
 content for EMAIL/IM.", "text/plain; charset=UTF-8")));
part2_mp.addBodyPart(part2_mp_partPlain);
MimeBodyPart part2_mp_partRich = new MimeBodyPart();
part2_mp_partRich.setDataHandler(new DataHandler(new
 StringDataSource("<html><head></head><body><b><i>" + "HTML content for EMAIL/IM."
 +
    "</i></b></body></html>", "text/html; charset=UTF-8")));
part2_mp.addBodyPart(part2_mp_partRich);
part2.setContent(part2_mp, part2_mp.getContentType());
part2.addHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "EMAIL");
part2.addHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "IM");
// add second part
```

```
mp.addBodyPart(part2);

// set the content of the message
message.setContent(new DataHandler(mp, mp.getContentType()));

// set the MultiplePayload flag to true
MimeHeader multiHeader = new MimeHeader();
multiHeader.setName(oracle.sdp.messaging.Message.HEADER_SDPM_MULTIPLE_PAYLOAD);
multiHeader.setValue(Boolean.TRUE.toString());
message.getHeaders().add(multiHeader);
```

62.3.2 API Reference for Interface Message

The API reference for interface Message can be accessed from the Javadoc.

62.3.3 API Reference for Enum DeliveryType

The API reference for enum `DeliveryType` can be accessed from the Javadoc.

62.3.4 Addressing a Message

This section describes type of addresses and how to create address objects.

62.3.4.1 Types of Addresses

There are two types of addresses, *device addresses* and *user addresses*. A device address can be of various types, such as email addresses, instant messaging addresses, and telephone numbers. User addresses are user IDs in a user repository.

62.3.4.2 Creating Address Objects

You can address senders and recipients of messages by using the class `MessagingFactory` to create `Address` objects defined by the `Address` interface.

62.3.4.2.1 Creating a Single Address Object Example 62–7 shows code for creating a single `Address` object:

Example 62–7 Creating a Single Address Object

```
Address recipient = MessagingFactory.createAddress("Email:john.doe@oracle.com");
```

62.3.4.2.2 Creating Multiple Address Objects in a Batch Example 62–8 shows code for creating multiple `Address` objects in a batch:

Example 62–8 Creating Multiple Address Objects in a Batch

```
String[] recipientsStr = {"Email:john.doe@oracle.com", "IM:john.doe@oracle.com"};
Address[] recipients = MessagingFactory.createAddress(recipientsStr);
```

62.3.4.2.3 Adding Sender or Recipient Addresses to a Message Example 62–9 shows code for adding sender or recipient addresses to a message:

Example 62–9 Adding Sender or Recipient Addresses to a Message

```
Address sender = MessagingFactory.createAddress("Email:john.doe@oracle.com");
Address recipient = MessagingFactory.createAddress("Email:jane.doe@oracle.com");
message.addSender(sender);
message.addRecipient(recipient);
```

62.3.4.3 Creating a Recipient with a Failover Address

Example 62–10 shows code for creating a recipient with a failover address:

Example 62–10 Creating a Single Address Object with Failover

```
String recipientWithFailoverStr = "Email:john.doe@oracle.com,
IM:john.doe@oracle.com";
Address recipient = MessagingFactory.createAddress(recipientWithFailoverStr);
```

62.3.4.4 Recipient Types

The WS API provides support for sending and receiving messages with To/Cc/Bcc recipients for use with the email driver:

- To send a message and specify a Cc/Bcc recipient, create the `oracle.ucs.messaging.ws.Address` object using `oracle.ucs.messaging.ws.MessagingFactory.buildAddress` method. The arguments are the address value (for example, user@domain.com), delivery type (for example, `DeliveryType.EMAIL`), and email mode (for example, "Cc" or "Bcc").
- To determine the recipient type of an existing address object, for example in a received message, use the `oracle.ucs.messaging.ws.MessagingFactory.getRecipientType` method, passing it the Address object. It returns a string indicating the recipient type.

62.3.4.5 API Reference for Class MessagingFactory

The API reference for class `MessagingFactory` can be accessed from the Javadoc.

62.3.4.6 API Reference for Interface Address

The API reference for interface `Address` can be accessed from the Javadoc.

62.3.5 User Preference Based Messaging

When sending a message to a user recipient (to leverage the user's messaging preferences), you can pass facts (current values) for various business terms in the message as metadata. The UMS server matches the supplied facts in the message against conditions for business terms specified in the user's messaging filters.

> **Note:** All facts must be added as metadata in the `oracle.sdp.messaging.Message.NAMESPACE_NOTIFICATION_PREFERENCES` namespace. Metadata in other namespaces are ignored (for resolving user messaging preferences).

Example 62–11 shows how to specify a user recipient and supply facts for business terms for the user preferences in a message. For a complete list of supported business terms, refer to Chapter 64, "User Messaging Preferences."

Example 62–11 User Preference Based Messaging

```
Message message = MessagingFactory.createMessage();
// create and add a user recipient
Address userRecipient1 = MessagingFactory.createAddress("USER:sampleuser1");
message.addRecipient(userRecipient1);
// specify business term facts
```

```
MessagingFactory.setMetadata(message, oracle.sdp.messaging.Message.NAMESPACE_
NOTIFICATION_PREFERENCES, "Customer Name", "ACME");
//  where "Customer Name" is the Business Term name, and "ACME" is the
Business Term value (i.e, fact).
```

62.4 Retrieving Message Status

After sending a message, you can use Oracle UMS to retrieve the message status either synchronously or asynchronously.

62.4.1 Synchronous Retrieval of Message Status

To perform a synchronous retrieval of current status, use the following flow from the `MessagingClient` API:

```
String messageId = messagingClient.send(message);
List<Status> statuses = messagingClient.getStatus(messageId, null)
```

or,

```
List<Status> statuses = messagingClient.getStatus(messageId, addresses) --- where
addresses is a "List<Address>" of one or more of the recipients set in the
message.
```

62.4.2 Asynchronous Receiving of Message Status

To receive statuses asynchronously, a client application must implement the Listener Web service as described in `listener.wsdl`. There is no constraint on how the listener endpoint must be implemented. For example, one method is to use the `javax.xml.ws.Endpoint` JAX-WS Service API to publish a web service endpoint. This mechanism is available in Java SE 6 and does not require the consumer to explicitly define a Java EE servlet module.

However, a servlet-based listener implementation is acceptable as well.

When sending a message, the client application can provide a reference to the listener endpoint, consisting of the endpoint URL and a SOAP interface name. As statuses are generated during the processing of the message, the UMS server invokes the listener endpoint' s `onStatus` method to notify the client application.

62.4.2.1 Creating a Listener Programmatically

Listeners are purely programmatic. You create a listener by implementing the `oracle.ucs.messaging.ws.Listener` interface. You can implement it as any concrete class - one of your existing classes, a new class, or an anonymous or inner class.

The following code example shows how to implement a status listener:

```
@PortableWebService(serviceName="ListenerService",
targetNamespace="http://xmlns.oracle.com/ucs/messaging/",
endpointInterface="oracle.ucs.messaging.ws.Listener",
wsdlLocation="META-INF/wsdl/listener.wsdl",
portName="Listener")
public class MyListener implements Listener {
  public MyListener() {
  }

  @Override
  public void onMessage(Message message, byte[] correlator) throws
```

```
MessagingException {
    System.out.println("I got a message!");
  }

  @Override
  public void onStatus(Status status, byte[] correlator) throws MessagingException
{
    System.out.println("I got a status!");
  }
}
```

62.4.2.2 Publish the Callback Service

When the To publish the callback service, you can either declare a servlet in web.xml in a web module within your application, or use the JAX-WS javax.xml.ws.Endpoint class's publish method to programmatically publish a WS endpoint (Example 62–12):

Example 62–12 Publish the Callback Service

```
Listener myListener = new MyListener();
String callbackURL = "http://host:port/umswscallback";
Endpoint myEndpoint = javax.xml.ws.Endpoint.publish(callbackURL, myListener);
```

62.4.2.3 Stop a Dynamically Published Endpoint

To stop a dynamically published endpoint, call the `stop()` method on the Endpoint object returned from `Endpoint.publish()` (Example 62–13).

Example 62–13 Stop a Dynamically Published Endpoint

```
// When done, stop the endpoint, ideally in a finally block or other reliable
cleanup mechanism
myEndpoint.stop();
```

62.4.2.4 Registration

Once the listener web service is published, you must register the fact that your client has such an endpoint. There are the following relevant methods in the MessagingClient API:

- `setStatusListener(ListenerReference listener)`
- `send(Message message, ListenerReference listener, byte[] correlator)`

`setStatusListener()` registers a "default" status listener whose callback is invoked for any incoming status messages. A listener passed to `send()` is only invoked for status updates related to the corresponding message.

62.5 Receiving a Message

This section describes how an application receives messages. To receive a message you must first register an access point. From the application perspective there are two modes for receiving a message, synchronous and asynchronous.

62.5.1 Registering an Access Point

The client application can create and register an access point, specifying that it wants to receive incoming messages sent to a particular address. When registering an access point, the client application can provide a reference to the listener endpoint, consisting of the endpoint URL and a SOAP interface name. As messages arrive, the UMS server invokes the listener endpoint' s `onMessage` method to notify the client application.

The client application can then invoke the receive method to fetch the pending messages. When receiving messages without specifying an access point, the application receives messages for any of the access points that it has registered. Otherwise, if an access point is specified, the application receives messages sent to that access point.

`AccessPoint` represents one or more device addresses to receive incoming messages. An application that wants to receive incoming messages must register one or more access points that represent the recipient addresses of the messages. The server matches the recipient address of an incoming message against the set of registered access points, and routes the incoming message to the application that registered the matching access point.

You can use `MessagingFactory.createAccessPoint` to create an access point and `MessagingClient.registerAccessPoint` to register it for receiving messages.

To register an SMS access point for the number `9000`:

```
AccessPoint accessPointSingleAddress =
 MessagingFactory.createAccessPoint(AccessPointType.SINGLE_ADDRESS,
 DeliveryType.SMS, "9000");
messagingClient.registerAccessPoint(accessPointSingleAddress);
```

To register SMS access points in the number range `9000` to `9999`:

```
AccessPoint accessPointRangeAddress =
 MessagingFactory.createAccessPoint(AccessPointType.NUMBER_RANGE,
 DeliveryType.SMS,"9000,9999");
messagingClient.registerAccessPoint(accessPointRangeAddress);
```

62.5.2 Synchronous Receiving

Receive is a nonblocking operation. If there are no pending messages for the application or access point, the call returns immediately with an empty list. Receive is not guaranteed to return all available messages, but may return only a subset of available messages for efficiency reasons.

You can use the method `MessagingClient.receive` to synchronously receive messages. This is a convenient polling method for light-weight clients that do not want the configuration overhead associated with receiving messages asynchronously. This method returns a list of messages that are immediately available in the application inbound queue.

It performs a nonblocking call, so if no message is currently available, the method returns null.

Note: A single invocation does not guarantee retrieval of all available messages. You must poll to ensure receiving all available messages.

62.5.3 Asynchronous Receiving

To receive messages asynchronously, a client application must implement the `Listener` web service as described in `listener.wsdl`. There is no constraint on how the listener endpoint must be implemented. For example, one mechanism is using the `javax.xml.ws.Endpoint` JAX-WS Service API to publish a Web service endpoint. This mechanism is available in Java SE 6 and does not require the consumer to explicitly define a Java EE servlet module. However, a servlet-based listener implementation is also acceptable.

62.5.3.1 Creating a Listener Programmatically

Listeners are purely programmatic. You create a listener by implementing the `oracle.ucs.messaging.ws.Listener` interface. You can implement it as any concrete class - one of your existing classes, a new class, or an anonymous or inner class.

The following code example shows how to implement a message listener:

```
@PortableWebService(serviceName="ListenerService",
targetNamespace="http://xmlns.oracle.com/ucs/messaging/",
endpointInterface="oracle.ucs.messaging.ws.Listener",
wsdlLocation="META-INF/wsdl/listener.wsdl",
portName="Listener")
public class MyListener implements Listener {
  public MyListener() {
  }

  @Override
  public void onMessage(Message message, byte[] correlator) throws
MessagingException {
    System.out.println("I got a message!");
  }

  @Override
  public void onStatus(Status status, byte[] correlator) throws MessagingException
{
    System.out.println("I got a status!");
  }
}
```

You pass a reference to the Listener object to the `setMessageListener` or `registerAccessPoint` methods, as described in "Default Message Listener" and "Per Access Point Message Listener". When a message arrives for your application, the UMS infrastructure invokes the Listener's `onMessage` method.

62.5.3.2 Default Message Listener

The client application typically sets a default message listener (Example 62–14). This listener is invoked for any delivery statuses for messages sent by this client application that do not have an associated listener. When Oracle UMS receives messages addressed to any access points registered by this client application, it invokes the `onMessage` callback for the client application's default listener.

To remove a default listener, call this method with a null argument.

Example 62–14 Default Message Listener

```
messagingClient.setMessageListener(new MyListener());
```

62.5.3.3 Per Access Point Message Listener

The client application can also register an access point and specify a `Listener` object and an optional correlator object (Example 62–15). When incoming messages arrive at the specified access point address, the specified listener's `onMessage` method is invoked. The originally-specified correlator object is also passed to the callback method.

Example 62–15 Per Access Point Message Listener

```
messagingClient.registerAccessPoint(accessPoint, new MyListener(), null);
```

62.5.4 Message Filtering

A `MessageFilter` is used by an application to exercise greater control over what messages are delivered to it. A `MessageFilter` contains a matching criterion and an action. An application can register a series of message filters; they are applied in order against an incoming (received) message; if the criterion matches the message, the action is taken. For example, an application can use `MessageFilters` to implement necessary blacklists, by rejecting all messages from a given sender address.

You can use `MessagingFactory.createMessageFilter` to create a message filter, and `MessagingClient.registerMessageFilter` to register it. The filter is added to the end of the current filter chain for the application. When a message is received, it is passed through the filter chain in order; if the message matches a filter's criterion, the filter's action is taken immediately. If no filters match the message, the default action is to accept the message and deliver it to the application.

For example, to reject a message with the subject `"spam"`:

```
MessageFilter subjectFilter = MessagingFactory.createMessageFilter("spam",
 FilterFieldType.SUBJECT, null, FilterActionType.REJECT);
messagingClient.registerMessageFilter(subjectFilter);
```

To reject messages from email address `spammer@foo.com`:

```
MessageFilter senderFilter =
 MessagingFactory.createBlacklistFilter("spammer@foo.com");
messagingClient.registerMessageFilter(senderFilter);
```

62.6 Configuring for a Cluster Environment

The API supports an environment where client applications and the UMS server are deployed in a cluster environment. For a clustered deployment to function as expected, client applications must be configured correctly. The following rules apply:

- Two client applications are considered to be instances of the same application if they use the same `ApplicationName` configuration parameter.
- The `ApplicationInstanceName` configuration parameter enables you to distinguish instances from one another.
- Application sessions are instance-specific. You can set the session flag on a message to ensure that any reply is received by the instance that sent the message.
- Listener correlators are instance-specific. If two different instances of an application register listeners and supply different correlators, then when instance A's listener is invoked, correlator A is supplied; when instance B's listener is invoked, correlator B is supplied.

62.7 Configuring Security

The following sections discuss security considerations:

- Section 62.7.1, "Client and Server Security"
- Section 62.7.2, "Listener/Callback Security"

62.7.1 Client and Server Security

There are two supported security modes for the UMS Web Service: Security Assertions Markup Language (SAML) tokens and username tokens.

The supported SAML-based policy is "oracle/wss11_saml_token_with_message_protection_client_policy". This policy establishes a trust relationship between the client application and the UMS server based on the exchange of cryptographic keys. The client application is then allowed to assert a user identity that is respected by the UMS server. To use SAML tokens for WS-Security, some keystore configuration is required for both the client and the server. See Example 62–2 for more details about configuring SAML security in a UMS Web service client.

The supported username token policy is "oracle/wss11_username_token_with_message_protection_client_policy". This policy passes an encrypted username/password token in the WS-Security headers, and the server authenticates the supplied credentials. It is highly recommended that the username and password be stored in the Credential Store, in which case only a Credential Store key must be passed to the MessagingClient constructor, ensuring that credentials are not hard-coded or stored in an unsecure manner. See Example 62–1 for more details about configuring SAML security in a UMS Web service client.

62.7.2 Listener/Callback Security

Username token and SAML token security are also supported for the Listener callback web services. When registering a listener, the client application must supply additional parameters specifying the security policy and any key or credential lookup information that the server requires to establish a secure connection.

Example 62–16 illustrates how to establish a secure callback endpoint using username token security:

Example 62–16 Establishing a Secure Callback Endpoint Using Username Token Security

```
MessagingClient client = new MessagingClient(clientParameters);
...
ListenerReference listenerRef = new ListenerReference();
// A web service implementing the oracle.ucs.messaging.ws.Listener
// interface must be available at the specified URL.
listenerRef.setEndpoint(myCallbackURL);
Parameter policyParam = new Parameter();
policyParam.setName(ClientConstants.POLICY_STRING);
policyParam.setValue("oracle/wss11_username_token_with_message_protection_client_
policy");
listenerRef.getParameters.add(policyParam);
// A credential store entry with the specified key must be
// provisioned on the server side so it will be available when the callback
// is invoked.
Parameter csfParam = new Parameter();
csfParam.setName(oracle.wsm.security.util.SecurityConstants.ClientConstants.WSS_
CSF_KEY);
```

```
csfParam.setValue("callback-csf-key");
listenerRef.getParameters.add(csfParam);
client.setMessageListener(listenerRef);
```

62.8 Threading Model

Instances of the WS MessagingClient class are not thread-safe due to the underlying services provided by the JAX-WS stack. You are responsible for ensuring that each instance is used by only one thread at a time.

62.9 Sample Chat Application with Web Services APIs

This chapter describes how to create, deploy and run the sample chat application with Web Services APIs provided with Oracle User Messaging Service on OTN.

Note: To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL:
`https://soasamples.samplecode.oracle.com/`

Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

Note: To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This chapter contains the following sections:

- Section 62.9.1, "Overview"
- Section 62.9.2, "Running the Pre-Built Sample"
- Section 62.9.3, "Testing the Sample"

62.9.1 Overview

This sample demonstrates how to create a web-based chat application to send and receive messages through email, SMS, or IM. The sample uses the Web Service APIs to interact with a User Messaging server. You define an application server connection in Oracle JDeveloper, and deploy and run the application.

The application is provided as a pre-built Oracle JDeveloper project that includes a simple Web chat interface.

62.9.1.1 Provided Files

The following files are included in the sample application:

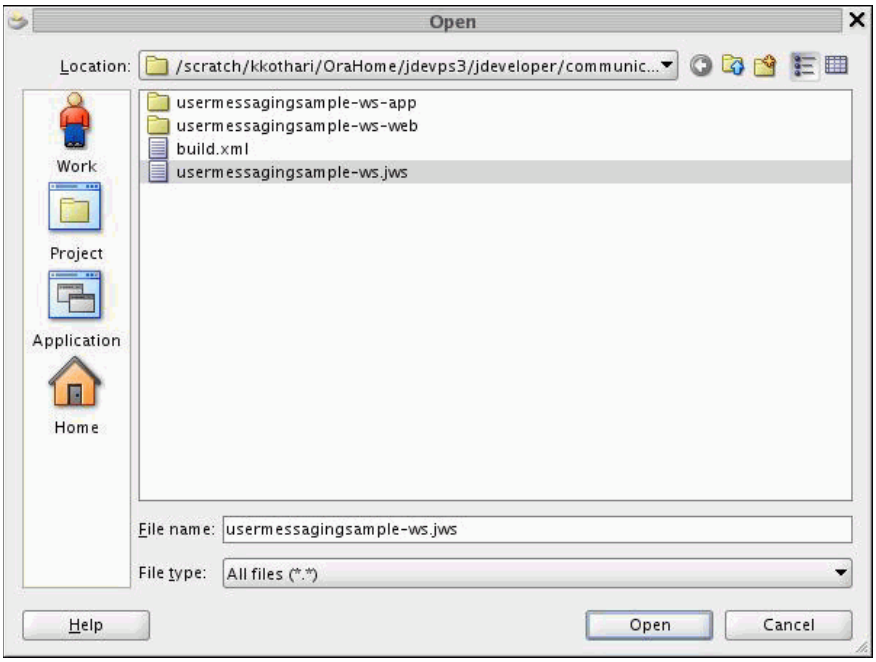
- usermessagingsample-ws-src.zip – the archive containing the source code and Oracle JDeveloper project files.
- usermessagingsample-ws.ear - the pre-built sample application that can be deployed to the container.

62.9.2 Running the Pre-Built Sample

Perform the following steps to run and deploy the pre-built sample application:

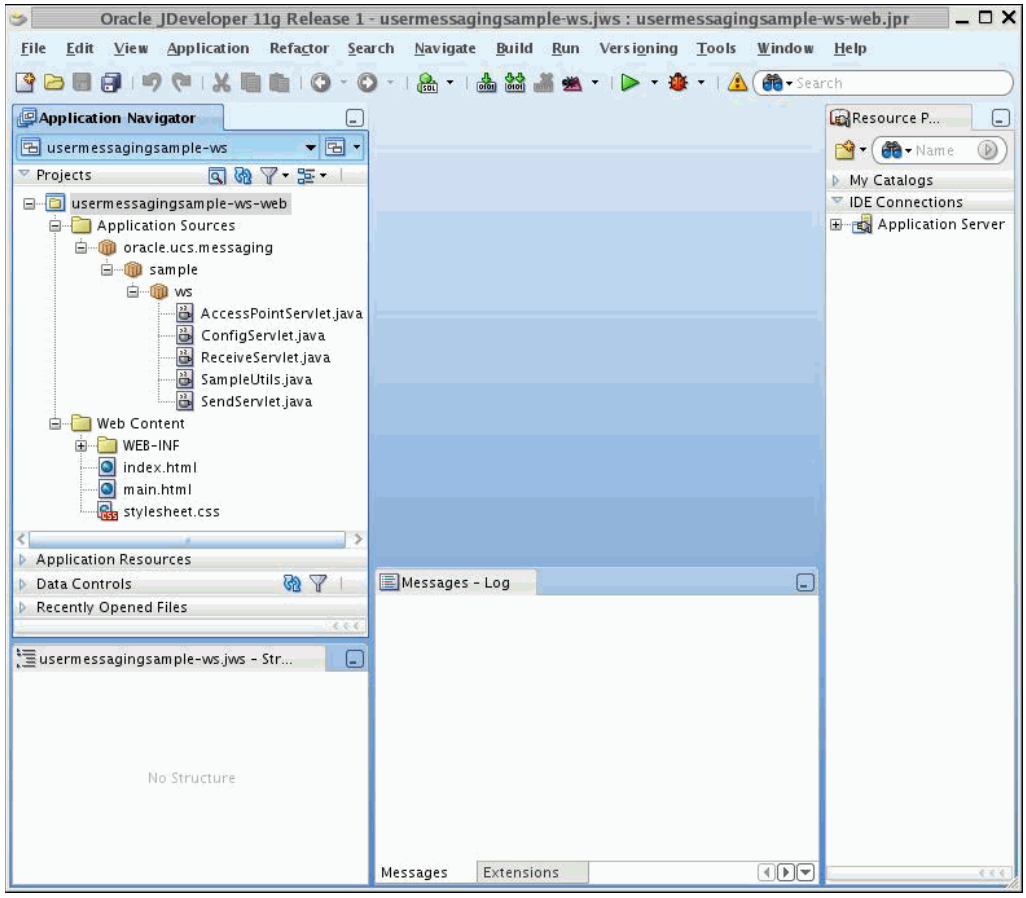
1. Extract "usermessagingsample-ws-src.zip" and open **usermessagingsample-ws.jws** in Oracle JDeveloper.

Figure 62–1 Opening the Project in Oracle JDeveloper



In the Oracle JDeveloper main window the project appears.

Figure 62–2 Oracle JDeveloper Main Window



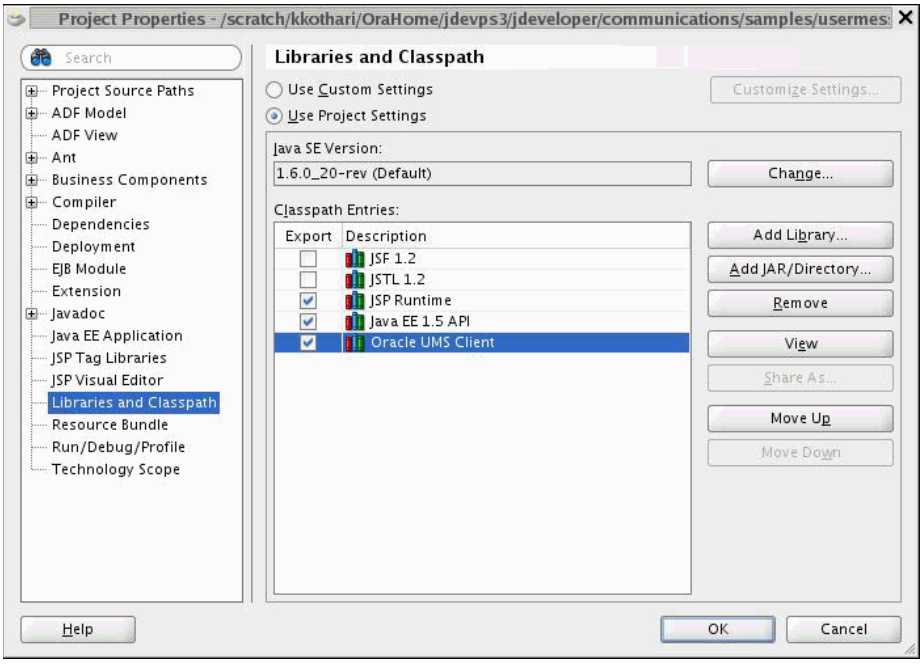
The application contains one Web module. All of the source code for the application is in place.

2. Satisfy the build dependencies for the sample application by ensuring the "Oracle UMS Client" library is used by the Web module.

    1. In the Application Navigator, right-click web module usermessagingsample-ws-war, and select **Project Properties**.

    2. In the left pane, select **Libraries and Classpath**.

Figure 62–3 Adding a Library

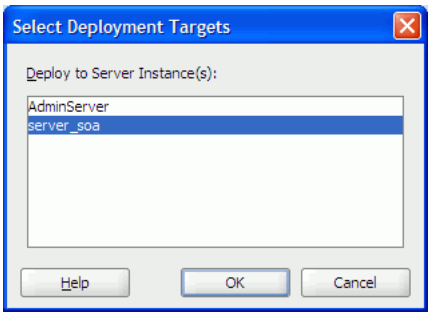


3. Click **OK**.

3. Create an Application Server Connection by right-clicking the project in the navigation pane and selecting **New**. Follow the instructions in Section 62.10, "Creating a New Application Server Connection".

4. Deploy the project by selecting the **usermessasgingsample-ws project**, **Deploy**, **usermessasgingsample-ws**, **to**, and **SOA_server** (Figure 62–4).

Figure 62–4 Deploying the Project



5. Verify that the message `Build Successful` appears in the log.

6. Verify that the message `Deployment Finished` appears in the deployment log.

   You have successfully deployed the application.

62.9.3 Testing the Sample

Perform the following steps to run and test the sample:

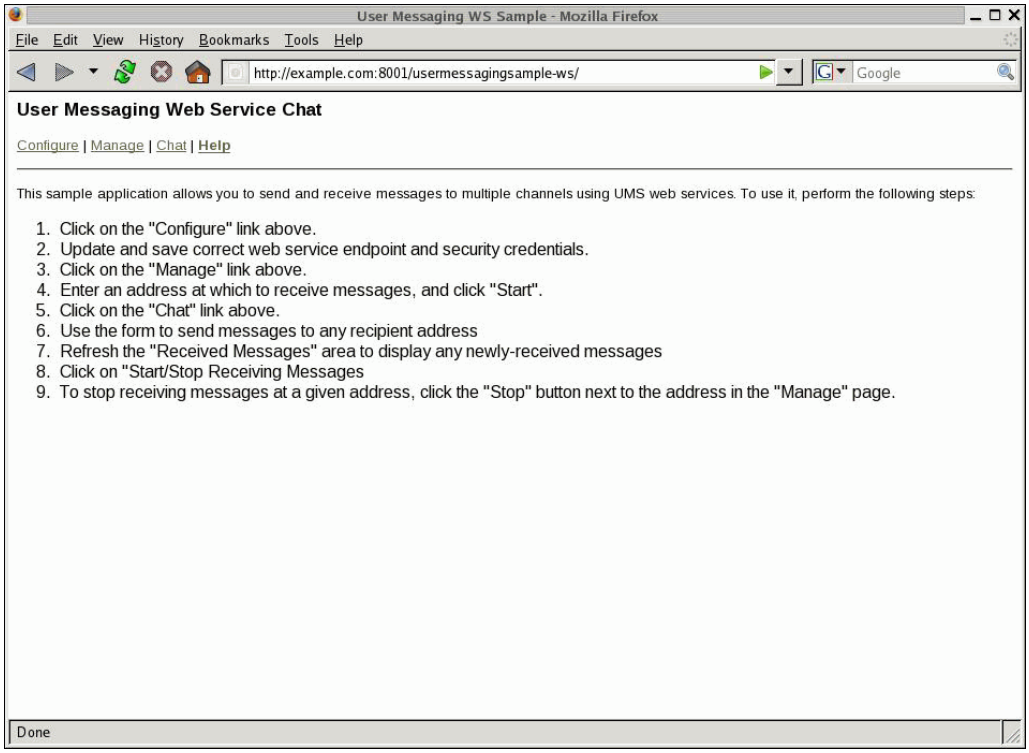
1. Open a web browser.

2. Navigate to the URL of the application as follows, and log in:

`http://host:port/usermessagingsample-ws/`

The Messaging Web Services Sample web page appears (Figure 62–5). This page contains navigation tabs and instructions for the application.

Figure 62–5 Messaging Web Services Sample Web Page

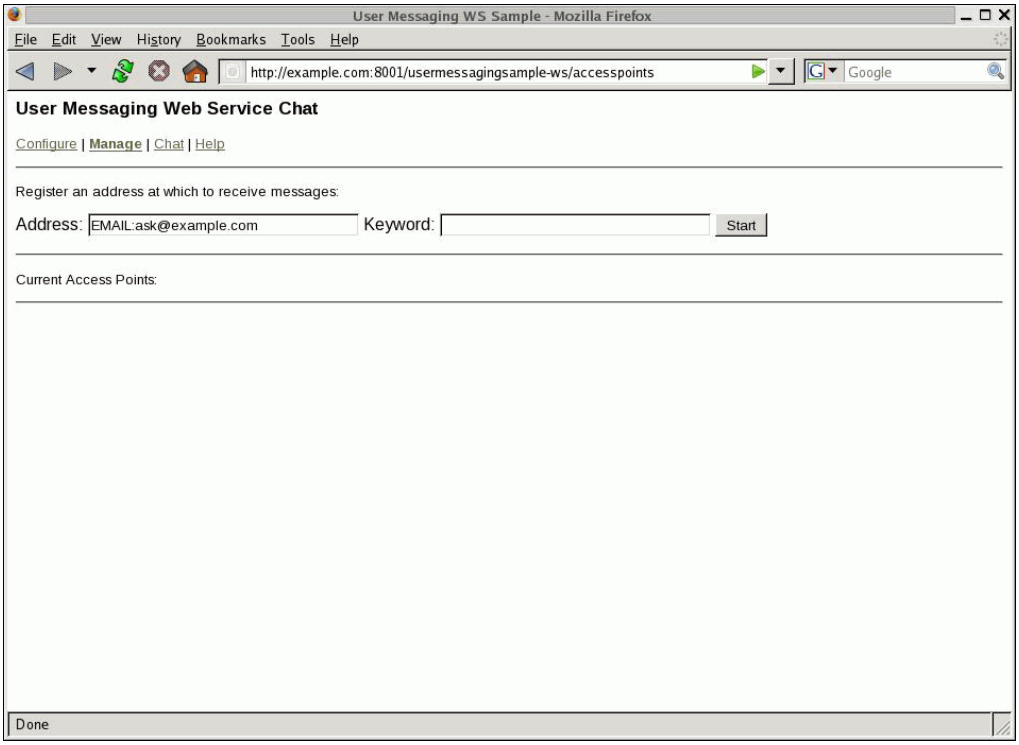


3. Click **Configure** and enter the following values (Figure 62–6):
   - Specify the Web Service endpoint. For example, `http://example.com:8001/ucs/messaging/webservice`
   - Specify the Username and Password.
   - Specify a Policy (required if the User Messaging Service instance has WS security enabled).

Figure 62–6 Configuring the Web Service Endpoints and Credentials

4. Click **Save**.
5. Click **Manage**.
6. Enter an address and optional keyword at which to receive messages (Figure 62–7).

Figure 62–7 Registering an Access Point

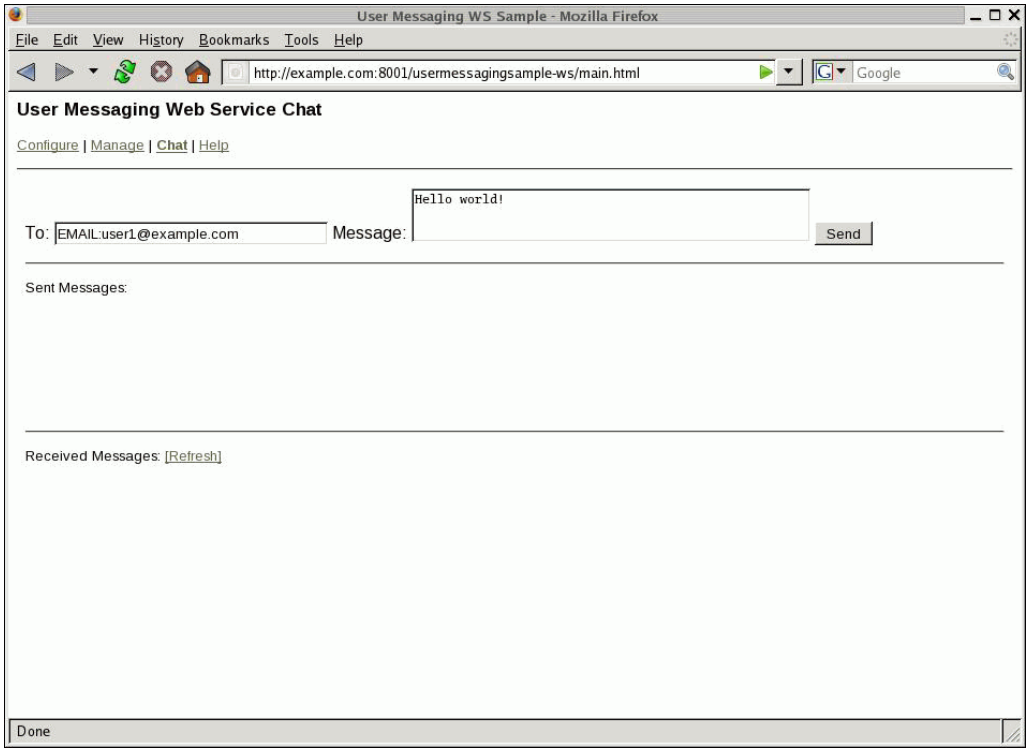


7. Click **Start**.

   Verify that the message `Registration operation succeeded` appears.
8. Click **Chat** (Figure 62–8).

9. Enter recipients in the **To:** field in the format illustrated in Figure 62–8.
10. Enter a message.
11. Click **Send**.
12. Verify that the message is received.

Figure 62–8 Running the Sample

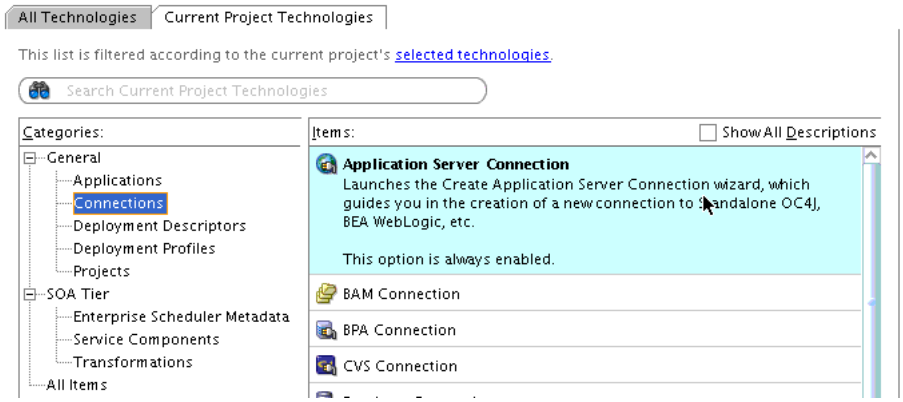


62.10 Creating a New Application Server Connection

Perform the following steps to create an Application Server Connection.

1. Create a new Application Server Connection by right-clicking the project and selecting **New**, **Connections**, and **Application Server Connection** (Figure 62–9).

Figure 62–9 New Application Server Connection



2. Name the connection `SOA_server` and click **Next** (Figure 62–10).
3. Select **WebLogic 10.3** as the **Connection Type**.

Figure 62–10 New Application Server Connection

4. Enter the authentication information. A typical value for user name is `weblogic`.
5. In the Connection dialog, enter the hostname, port, and SSL port for the SOA admin server, and enter the name of the domain for WLS Domain.
6. Click **Next**.
7. In the Test dialog, click **Test Connection**.
8. Verify that the message `Success!` appears.

   The Application Server Connection has been created.

63

Parlay X Web Services Multimedia Messaging API

This chapter describes the Parlay X Multimedia Messaging Web Service that is available with Oracle User Messaging Service and how to use the Parlay X Web Services Multimedia Messaging API to send and receive messages through Oracle User Messaging Service.

> **Note:** To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This chapter includes the following sections:

> **Note:** To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL:
> `https://soasamples.samplecode.oracle.com/`
>
> Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

> **Note:** Oracle User Messaging Service also ships with a Java client library that implements the Parlay X API.

63.1 Introduction to Parlay X Messaging Operations

The following sections describe the semantics of each of the supported operations along with implementation-specific details for the Parlay X Gateway. The following

tables, describing input/output message parameters for each operation, are taken directly from the Parlay X specification.

Oracle User Messaging Service implements a subset of the Parlay X 2.1 Multimedia Messaging specification. Specifically Oracle User Messaging Service supports the SendMessage and ReceiveMessage interfaces. The MessageNotification and MessageNotificationManager interfaces are not supported.

63.2 Send Message Interface

The SendMessage interface enables you to send a message to one or more recipient addresses by using the `sendMessage` operation, or get the delivery status for a previously sent message by using the getMessageDeliveryStatus operation. The following requirements apply:

- A recipient address must conform to the address format requirements of Oracle User Messaging Service (in addition to being a valid URI). The general format is *`delivery_type`*`:`*`protocol_specific_address`*, such as `email:user@domain`, `sms:5551212` or `im:user@jabberdomain`.
- Certain characters are not allowed in URIs; if it is necessary to include them in an address they can be encoded or escaped. Refer to the JavaDoc for `java.net.URI` for details on how to create a properly encoded URI.
- While the WSDL specifies that sender addresses can be any string, Oracle User Messaging Service requires that they be valid Messaging addresses.
- Oracle User Messaging Service requires that you specify sender addresses on a per-delivery type basis. So for a sender address to apply to a recipient of a given delivery type, say EMAIL, the sender address must also have delivery type of EMAIL. Since this operation allows multiple recipient addresses but only one sender address, the sender address only applies to the recipients with the same delivery type.
- Oracle User Messaging Service does not support the MessageNotification interface, and therefore do not produce delivery receipts, even if a receiptRequest is specified. In other words, the receiptRequest parameter is ignored.

63.2.1 sendMessage Operation

Table 63–1 describes message descriptions for the `sendMessageRequest` input in the `sendMessage` operation.

Table 63–1 sendMessage Input Message Descriptions

Part Name	Part Type	Optional	Description
addresses	xsd:anyURI[0..unbounded]	No	Destination address for this Message.
senderAddress	xsd:string	Yes	Message sender address. This parameter is not allowed for all 3rd party providers. The Parlay X server must handle this according to a SLA for the specific application and its use can therefore result in a PolicyException.

Table 63–1 (Cont.) sendMessage Input Message Descriptions

Part Name	Part Type	Optional	Description
subject	xsd:string	Yes	Message subject. If mapped to SMS, this parameter is used as the senderAddress, even if a separate senderAddress is provided.
priority	MessagePriority	Yes	Priority of the message. If not present, the network assigns a priority based on the operator policy.Charging to apply to this message.
charging	common: ChargingInformation	Yes	Charging to apply to this message.
receiptRequest	common:SimpleReference	Yes	Defines the application endpoint, interface name and correlator that is used to notify the application when the message has been delivered to a terminal or if delivery is impossible.

Table 63–2 describes `sendMessageResponse` output messages for the `sendMessage` operation.

Table 63–2 sendMessageResponse Output Message Descriptions

Part Name	Part Type	Optional	Description
result	xsd:string	No	This correlation identifier is used in a getMessageDeliveryStatus operation invocation to poll for the delivery status of all sent messages.

63.2.2 getMessageDeliveryStatus Operation

The `getMessageDeliveryStatus` operation gets the delivery status for a previously sent message. The input "requestIdentifier" is the "result" value from a sendMessage operation. This is the same identifier that is referred to as a Message ID in other Messaging documentation.

Table 63–3 describes the `getMessageDeliveryStatusRequest` input messages for the `getMessageDeliveryStatus` operation.

Table 63–3 getMessageDeliveryStatusRequest Input Message Descriptions

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	Identifier related to the delivery status request.

Table 63–4 describes the getMessageDeliveryStatusResponse output messages for the `getMessageDeliveryStatus` operation.

Table 63–4 getMessageDeliveryStatusResponse Output Message Descriptions

Part Name	Part Type	Optional	Description
result	DeliveryInformation [0..unbounded]	Yes	An array of status of the messages that were previously sent. Each array element represents a sent message, its destination address and its delivery status.

63.3 Receive Message Interface

The ReceiveMessage interface has three operations. The `getReceivedMessages` operation polls the server for any messages received since the last invocation of `getReceivedMessages`. Note that `getReceivedMessages` does not necessarily return any message content; it generally only returns message metadata.

The other two operations, `getMessage` and `getMessageURIs`, are used to retrieve message content.

63.3.1 getReceivedMessages Operation

This operation polls the server for any received messages. Note the following requirements:

- The registration ID parameter is a string that identifies the endpoint address for which the application wants to receive messages. See the discussion of the ReceiveMessageManager interface for more details.
- The Parlay X specification says that if the registration ID is not specified, all messages for this application should be returned. However, the WSDL says that the registration ID parameter is mandatory. Therefore our implementation treats the empty string ("") as the "not-specified" value. If you call getReceivedMessages with the empty string as your registration ID, you get all messages for this application. Therefore the empty string is not an allowed value of registration ID when calling startReceiveMessages.
- According to the Parlay X specification, if the received message content is "pure ASCII text", then the message content is returned inline within the MessageReference object, and the messageIdentifier (Message ID) element is null. Our implementation treats any content with Content-Type "text/plain", and with encoding "us-ascii" as "pure ASCII text" for the purposes of this operation. As per the MIME specification, if no encoding is specified, "us-ascii" is assumed, and if no Content-Type is specified, "text/plain" is assumed.
- The priority parameter is currently ignored.

Table 63–5 describes the `getReceivedMessagesRequest` input messages for the `getReceivedMessages` operation.

Table 63–5 getReceivedMessagesRequest Input Message Descriptions

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	Identifies the off-line provisioning step that enables the application to receive notification of Message reception according to the specified criteria.

Table 63–5 (Cont.) getReceivedMessagesRequest Input Message Descriptions

Part Name	Part Type	Optional	Description
priority	MessagePriority	Yes	The priority of the messages to poll from the Parlay X gateway. All messages of the specified priority and higher are retrieved. If not specified, all messages shall be returned, that is, the same as specifying "Low."

Table 63–6 describes the `getReceivedMessagesResponse` output messages for the `getReceivedMessages` operation.

Table 63–6 getReceivedMessagesResponse Output Message Descriptions

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	Identifies the off-line provisioning step that enables the application to receive notification of Message reception according to the specified criteria.
priority	MessagePriority	Yes	The priority of the messages to poll from the Parlay X gateway. All messages of the specified priority and higher are retrieved. If not specified, all messages shall be returned. This is equal to specifying Low.

63.3.2 getMessage Operation

The `getMessage` operation retrieves message content, using a message ID from a previous invocation of getReceivedMessages. There is no SOAP body in the response message; the content is returned as a single SOAP attachment.

Table 63–7 describes the `getMessageRequest` input messages for the `getMessage` operation.

Table 63–7 getMessageRequest Input Message Descriptions

Part Name	Part Type	Optional	Description
messageRefIdentifier	xsd:string	No	The identity of the message.

There are no `getMessageResponse` output messages for the `getMessage` operation.

63.3.3 getMessageURIs Operation

The `getMessageURIs` retrieves message content as a list of URIs. Note the following requirements:

- These URIs are HTTP URLs that can be dereferenced to retrieve the content.
- If the inbound message has a Content-Type of "multipart", then there are multiple URIs returned, one per subpart. If the Content-Type is not "multipart", then a single URI are returned.
- Per the Parlay X specification, if the inbound messages a body text part, defined as "the message body if it is encoded as ASCII text", it is returned inline within the

MessageURI object. For the purposes of our implementation, you define this behavior as follows:

- If the message's Content-Type is "text/*" (any text type), and if the charset parameter is "us-ascii", then the content is returned inline in the MessageURI object. There are no URIs returned since there is no content other than what is returned inline.
- If the message's Content-Type is "multipart/" (any multipart type), and if the first body part's Content-Type is "text/" with charset "us-ascii", then that part is returned inline in the MessageURI object, and there are no URIs returned corresponding to that part.
- Per the MIME specification, if the charset parameter is omitted, the default value of "us-ascii" is assumed. If the Content-Type header is not specified for the message, then a Content-Type of "text/plain" is assumed.

Table 63–8 describes the `getMessageURIsRequest` input messages for the `getMessageURIs` operation.

Table 63–8 getMessageURIsRequest Input Message Descriptions

Part Name	Part Type	Optional	Description
messageRefIdentifier	xsd:string	No	The identity of the message to retrieve.

Table 63–9 describes the `getMessageURIsResponse` output messages for the `getMessageURIs` operation.

Table 63–9 getMessageURIsResponse Output Message Descriptions

Part Name	Part Type	Optional	Description
result	MessageURI	No	Contains the complete message, consisting of the textual part of the message, if such exists, and a list of file references for the message attachments, if any.

63.4 Oracle Extension to Parlay X Messaging

The Parlay X Messaging specification leaves certain parts of the messaging flow undefined. The main area that is left undefined is the process for binding a client to an address for synchronous receiving (through the ReceiveMessage interface).

Oracle User Messaging Service includes an extension interface to Parlay X to support this process. The extension is implemented as a separate WSDL in an Oracle XML namespace to indicate that it is not an official part of Parlay X. Clients can choose to not use this additional interface or use it in some modular way such that their core messaging logic remains fully compliant with the Parlay X specification.

63.4.1 ReceiveMessageManager Interface

ReceiveMessageManager is the Oracle-specific interface for managing client registrations for receiving messages. Clients use this interface to start and stop receiving messages at a particular address. (This is analogous to the concept of registering/unregistering access points in the Messaging API).

63.4.1.1 startReceiveMessage Operation

Invoking this operation allows a client to bind itself to a given endpoint for receiving messages. Note the following requirements:

- An endpoint consists of an address and an optional "criteria", defined by the Parlay X specification as the first white space-delimited token of the message subject or content.
- In addition to the endpoint information, the client also specifies a "registration ID" when invoking this operation; this ID is just a unique string which can be used later to refer to this particular binding in the `stopReceiveMessage` and `getReceivedMessages` operations.
- If an endpoint is already registered by another client application, or the registration ID is already being used, a Policy Error results.
- Certain characters are not allowed in URIs; if it is necessary to include them in an address they can be encoded/escaped. See the javadoc for java.net.URI for details on how to create a properly encoded URI. For example, when registering to receive XMPP messages you must specify an address such as `IM:jabber|user@example.com`, however the pipe (|) character is not allowed in URIs, and must be escaped before submitting to the server.
- There is no guarantee that the server can actually receive messages at a given endpoint address. That depends on the overall configuration of Oracle User Messaging Service, particularly the Messaging drivers that are deployed in the system. No error is indicated if a client binds to an address where the server cannot receive messages.

The `startReceiveMessage` operation has the following inputs and outputs:

Table 63–10 describes the `startReceiveMessageRequest` input messages for the `startReceiveMessage` operation.

Table 63–10 startReceiveMessageRequest Input Message Descriptions

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	A registration identifier.
messageService ActivationNumber	xsd:anyURI	No	Message Service Activation Number.
criteria	xsd:string	Yes	Descriptive string.

There are no `startReceiveMessageResponse` output messages for the `startReceiveMessage` operation.

63.4.1.2 stopReceiveMessage Operation

Invoking this operation removes the previously-established binding between a client and a receiving endpoint. The client specifies the same registration ID that was supplied when `startReceiveMessage` was called to identify the endpoint binding that is being broken. If there is no corresponding registration ID binding known to the server for this application, a Policy Error results.

Table 63–11 describes the `stopReceiveMessageRequest` input messages for the `stopReceiveMessage` operation.

Table 63–11 stopReceiveMessageRequest Input Message Descriptions

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	A registration identifier.

There are no `stopReceiveMessageResponse` output messages for the `stopReceiveMessage` operation.

63.5 Parlay X Messaging Client API and Client Proxy Packages

While it is possible to assemble a Parlay X Messaging Client using only the Parlay X WSDL files and a web service assembly tool, prebuilt web service stubs and interfaces are provided for the supported Parlay X Messaging interfaces. Due to difficulty in assembling a web service with SOAP attachments in the style mandated by Parlay X, Oracle recommends the use of the provided API rather than starting from WSDL.

For a complete listing of the classes available in the Parlay X Messaging API, see the Messaging JavaDoc. The main entry points for the API are through the following client classes:

- `oracle.sdp.parlayx.multimedia_messaging.send.SendMessageClient`
- `oracle.sdp.parlayx.multimedia_messaging.receive.ReceiveMessageClient`
- `oracle.sdp.parlayx.multimedia_messaging.extension.receive_manager.ReceiveMessageManager`

Each client class allows a client application to invoke the operations in the corresponding interface. Additional web service parameters such as the remote gateway URL and any required security credentials, are provided when an instance of the client class is constructed. See the Javadoc for more details. The security credentials are propagated to the server using standard WS-Security headers, as mandated by the Parlay X specification.

The general process for a client application is to create one of the client classes above, set the necessary configuration items (endpoint, username, password), then invoke one of the business methods (for example, `SendMessageClient.sendMessage()`, and so on). For examples of how to use this API, see the Messaging samples on Oracle Technology Network (OTN), and specifically `usermessagingsample-parlayx-src.zip`.

63.6 Sample Chat Application with Parlay X APIs

This chapter describes how to create, deploy and run the sample chat application with Parlay X APIs provided with Oracle User Messaging Service on OTN.

Note: To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL: `https://soasamples.samplecode.oracle.com/`

Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

Note: To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This chapter contains the following sections:

63.6.1 Overview

This sample demonstrates how to create a web-based chat application to send and receive messages through email, SMS, or IM. The sample uses standards-based Parlay X Web Service APIs to interact with a User Messaging server. The sample application includes web service proxy code for each of three web service interfaces: the SendMessage and ReceiveMessage services defined by Parlay X, and the ReceiveMessageManager service which is an Oracle extension to Parlay X. You define an application server connection in Oracle JDeveloper, and deploy and run the application.

The application is provided as a pre-built Oracle JDeveloper project that includes a simple web chat interface.

63.6.1.1 Provided Files

The following files are included in the sample application:

- Project – the directory containing the archived Oracle JDeveloper project files.
- Readme.txt.
- Release notes

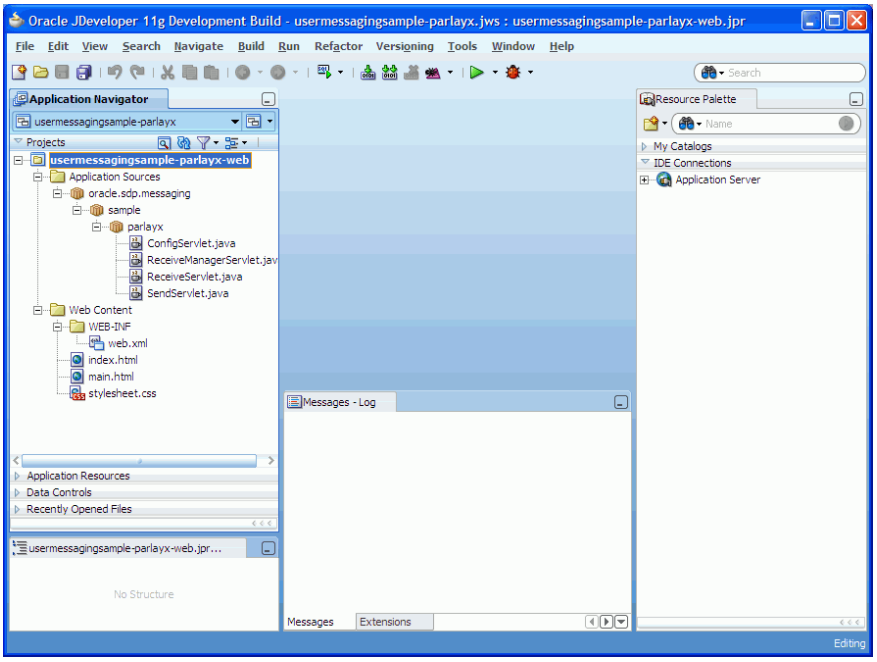
63.6.2 Running the Pre-Built Sample

Perform the following steps to run and deploy the pre-built sample application:

1. Open the **usermessagingsample-parlayx.jws** (contained in the .zip file) in Oracle JDeveloper.

   In the Oracle JDeveloper main window the project appears.

Figure 63–1 Oracle JDeveloper Main Window



2. In Oracle JDeveloper, select **File** > **Open**..., then navigate to the directory above and open workspace file "usermessagingsample-parlayx.jws".

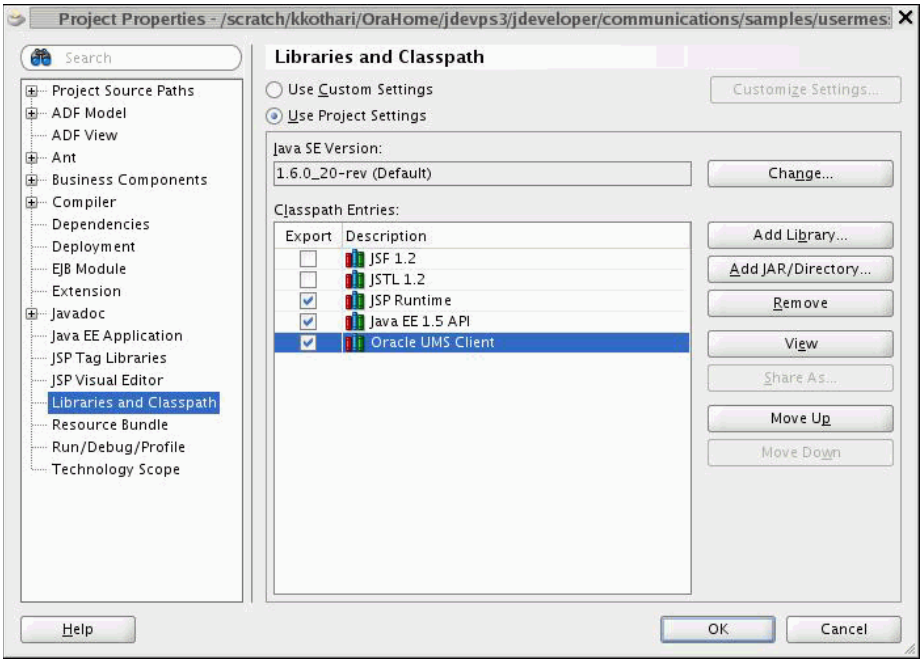
   This opens the precreated JDeveloper application for the Parlay X sample application. The application contains one web module. All of the source code for the application is in place. You must configure the parameters that are specific to your installation.

3. Satisfy the build dependencies for the sample application by ensuring the "Oracle UMS Client" library is used by the Web module.

   1. In the Application Navigator, right-click web module usermessagingsample-parlayx-war, and select **Project Properties**.

   2. In the left pane, select **Libraries and Classpath**.

Figure 63–2 Adding a Library

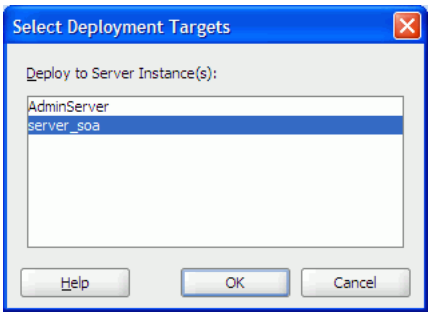


3. Click **OK**.

4. Create an Application Server Connection by right-clicking the project in the navigation pane and selecting New. Follow the instructions in Section 63.6.4, "Creating a New Application Server Connection".

5. Deploy the project by selecting the **usermessasgingsample-parlayx project**, **Deploy**, **usermessasgingsample-parlayx**, **to**, and **SOA_server** (Figure 63–3).

Figure 63–3 Deploying the Project



6. Verify that the message `Build Successful` appears in the log.

7. Enter the default revision and click **OK**.

8. Verify that the message `Deployment Finished` appears in the deployment log.

   You have successfully deployed the application.

   Before you can run the sample you must configure any additional drivers in Oracle User Messaging Service and configure a default device for the user receiving the message in User Messaging Preferences, as described in the following sections.

Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

63.6.3 Testing the Sample

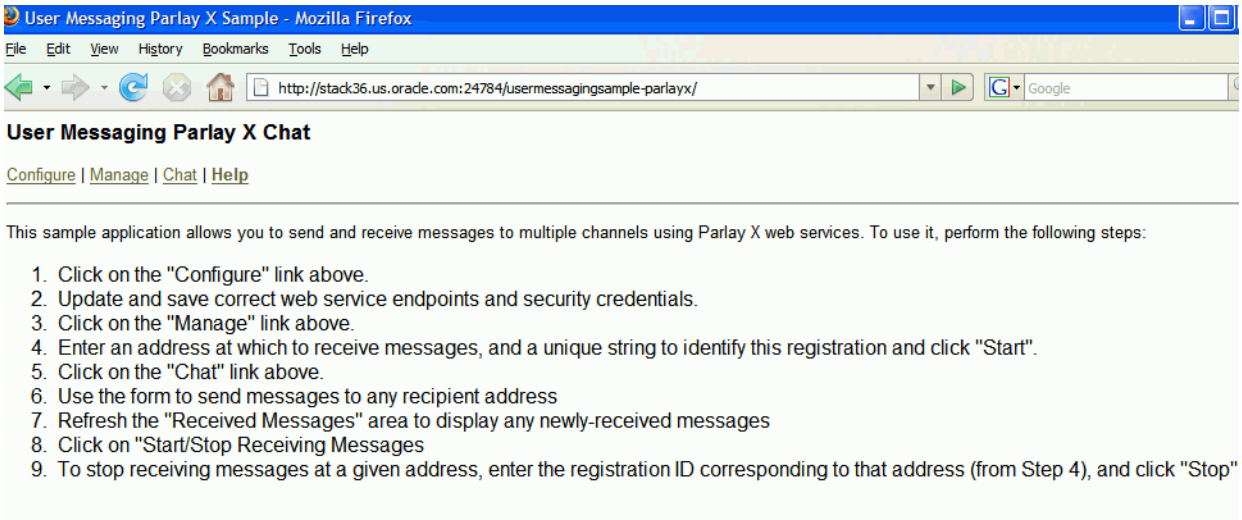
Perform the following steps to run and test the sample:

1. Open a web browser.
2. Navigate to the URL of the application as follows, and log in:

   `http://host:port/usermessagingsample-parlayx/`

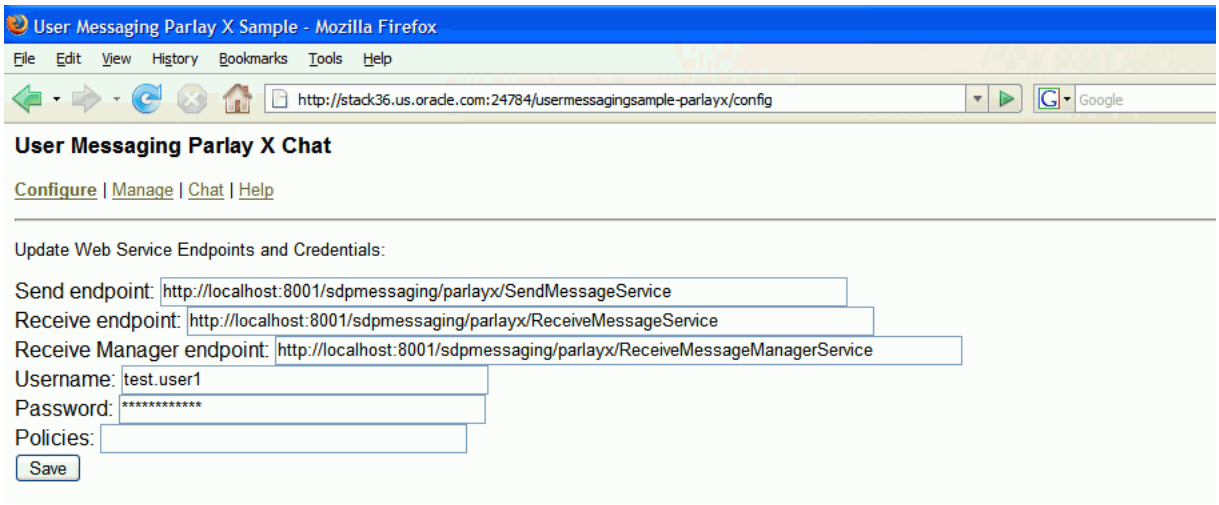
   The Messaging Parlay X Sample web page appears (Figure 63–4). This page contains navigation tabs and instructions for the application.

Figure 63–4 Messaging Parlay X Sample Web Page



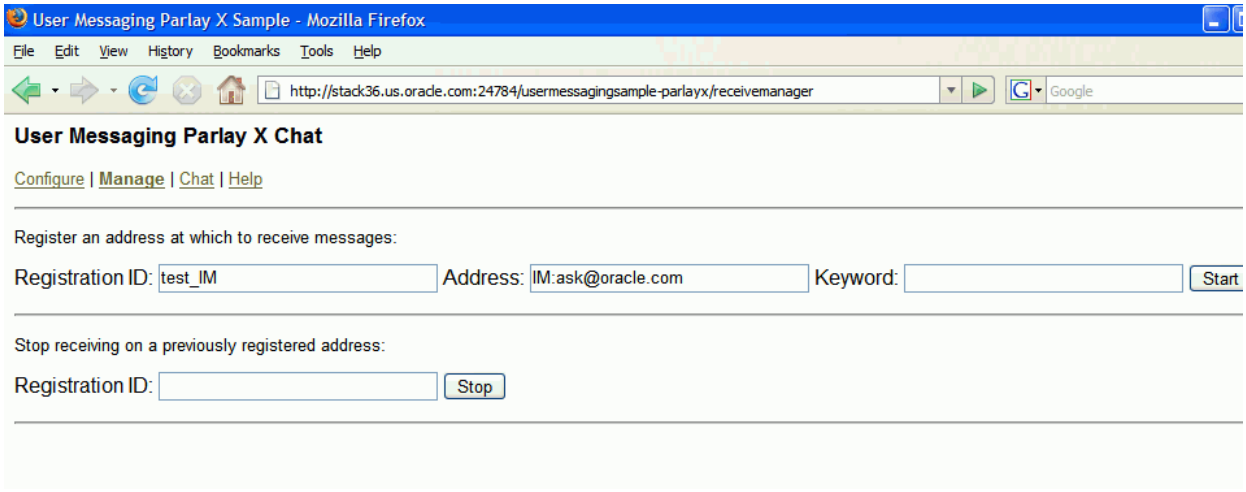
3. Click **Configure** and enter the following values (Figure 63–5):
   - Specify the Send endpoint. For example, `http://localhost:port/sdpmessaging/parlayx/SendMessageService`
   - Specify the Receive endpoint. For example, `http://localhost:port/sdpmessaging/parlayx/ReceiveMessageService`
   - Specify the Receive Manager endpoint. For example, `http://localhost:port/sdpmessaging/parlayx/ReceiveMessageMessageService`
   - Specify the Username and Password.
   - Specify a Policy (required if the User Messaging Service instance has WS security enabled).

Figure 63–5 Configuring the Web Service Endpoints and Credentials



4. Click **Save**.
5. Click **Manage**.
6. Enter a Registration ID to specify the registration and address at which to receive messages (Figure 63–6). You can also use this page to stop receiving messages at an address.

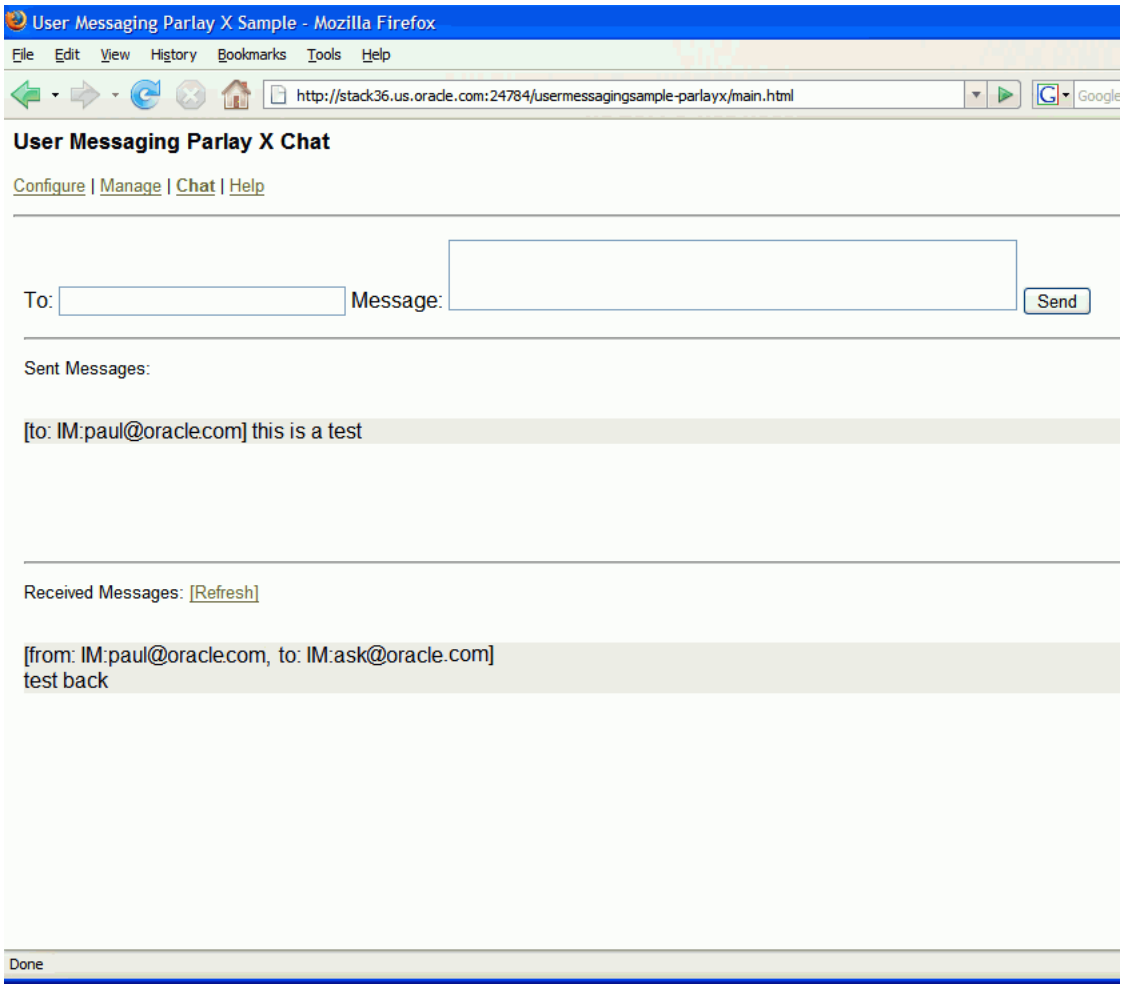
Figure 63–6 Specifying a Registration ID



7. Click **Start**.

   Verify that the message `Registration operation succeeded` appears.
8. Click **Chat** (Figure 63–7).
9. Enter recipients in the **To:** field in the format illustrated in Figure 63–7.
10. Enter a message.
11. Click **Send**.
12. Verify that the message is received.

Figure 63–7 Running the Sample

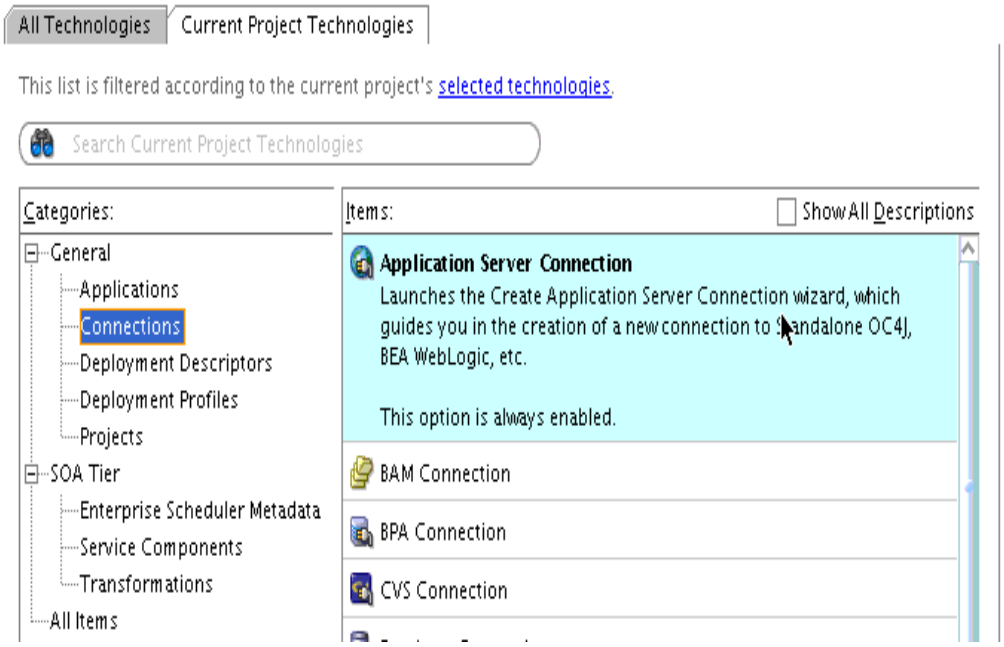


63.6.4 Creating a New Application Server Connection

Perform the following steps to create an Application Server Connection.

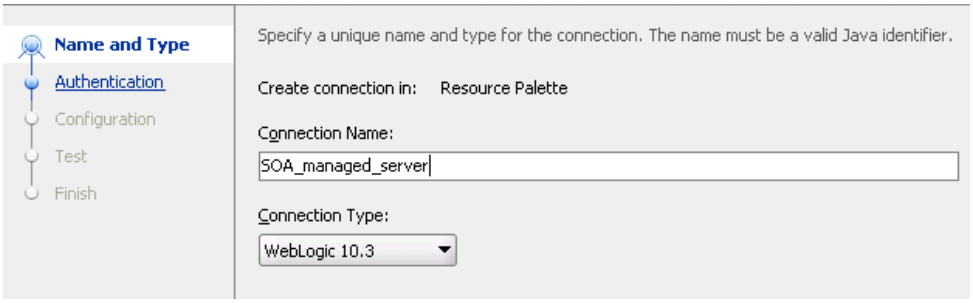
1. Create a new Application Server Connection by right-clicking the project and selecting **New**, **Connections**, and **Application Server Connection** (Figure 63–8).

Figure 63–8 New Application Server Connection



2. Name the connection `SOA_server` and click **Next** (Figure 63–9).
3. Select **WebLogic 10.3** as the **Connection Type**.

Figure 63–9 New Application Server Connection



4. Enter the authentication information. The typical value for username is `weblogic`.
5. In the Connection dialog, enter the hostname, port and SSL port for the SOA admin server, and enter the name of the domain for the Oracle WebLogic Server Domain.
6. Click **Next**.
7. On the Test dialog, click **Test Connection**.
8. Verify that the message `Success!` appears.

   The Application Server Connection has been created.

64

User Messaging Preferences

This chapter describes the User Messaging Preferences that are packaged with Oracle User Messaging Service. It describes how to work with messaging channels and to create contact rules using messaging filters.

> **Note:** To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This chapter includes the following sections:

64.1 Introduction to User Messaging Preferences

User Messaging Preferences allows a user who has access to multiple channels (delivery types) to control how, when, and where they receive messages. Users define filters, or delivery preferences, that specify which channel a message should be delivered to, and under what circumstances. Information about a user's devices and filters are stored in any database supported for use with Oracle Fusion Middleware.

For an application developer, User Messaging Preferences provide increased flexibility. Rather than an application needing business logic to decide whether to send an email or SMS message, the application can just send to the user, and the message is delivered according to the user's preferences.

Since preferences are stored in a database, this information is shared across all instances of User Messaging Preferences in a domain.

The `oracle.sdp.messaging.userprefs` package contains the User Messaging Preferences API classes. For more information, refer to the Javadoc.

64.1.1 Terminology

User Messaging Preferences defines the following terminology:

- Channel: the transport type, for example, email, voice, or SMS. Also, generally, a physical channel, such as a phone, or PDA.
- Channel address: one of the addresses that a channel can communicate with.

- Filters: a set of notification delivery preferences.
- System term: a pre-defined business term that cannot be extended by the administrator.
- Business term: a rule term defined and managed by the system administrator through Enterprise Manager. Business terms can be added, defined, or deleted.
- Rule term: a system term or a business term.
- Operators: comparison operators *equals*, *does not equal*, *contains*, or *does not contain*.
- Facts: data passed in from the message to be evaluated, such as *time sent*, or *sender*.
- Rules Engine: the User Messaging Preferences component that processes and evaluates filters.
- Comparison: a rule term and the associated comparison operator.
- Action: the action to be taken if the specified conditions in a rule are true, such as *Broadcast to All*, *Failover*, or *Do not Send to Any Channel*.

64.1.2 Configuration of Notification Delivery Preferences

User Messaging Preferences allows configuration of notification delivery preferences based on the following:

- a set of well-defined rule terms (system terms or business terms)
- a set of channel and the corresponding addresses supported by Oracle User Messaging Service
- a set of User Messaging Preferences filters that are transparently handled by a rules engine

One use case for notification delivery preference is for bugs entered into a bug tracking system. For example, user *Alex* wants to be notified through SMS and EMAIL channels for bugs filed against his product with priority = 1 by a customer type = Premium. For all other bugs with priority > 1, he only wants to be notified by EMAIL. Alex's preferences can be stated as follows:

Example 64–1 Notification Delivery Preferences

```
Rule (1): if (Customer Type = Premium) AND (priority = 1) then notify [Alex] using
 SMS and EMAIL.

Rule (2): if (Customer Type = Premium) AND (priority > 1) then notify [Alex] using
EMAIL.
```

A runtime service, the Oracle Rules Engine, evaluates the filters to process the notification delivery of user requests.

64.1.3 Delivery Preference Rules

A delivery preference rule consists of *rule comparisons* and *rule actions*. A rule comparison consists of a *rule term* (a system term or a business term) and the associated comparison operators. A rule action is the action to be taken if the specified conditions in a rule are true.

64.1.3.1 Data Types

Table 64–2 lists data types supported by User Messaging Preferences. Each system term and business term must have an associated data type, and each data type has a

set of pre-defined comparison operators. Administrators cannot extend these operators.

Table 64–1 Data Types Supported by User Messaging Preferences

Data Type	Comparison Operators	Supported Values
Date	<, >, between, <=, >=	Date is accepted as a `java.util.Date` object or string representing the number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT (in essence, the value from `java.util.Date.getTime()` or `java.util.Calendar.getTime()`).
Time	==, !=, between	A 4-digit integer to represent time of the day in HHMM format. First 2-digit is the hour in 24-hour format. Last 2-digit is minutes.
Number (Decimal)	<, >, between, <=, >=, isMultipleOf, isNotMultipleOf	A `java.lang.Double` object or a string representing a floating decimal point number with double precision.
String	==, !=, contains, not contains	Any arbitrary string.

Note: The String data type does not support regular expressions.

The Time data type is only available to System Terms.

64.1.3.2 System Terms

Table 64–2 lists system terms, which are pre-defined business terms. Administrators cannot extend the system terms.

Table 64–2 System Terms Supported by User Messaging Preferences

System Term	Data Type	Supported Values
Date	Date	Date is accepted as a `java.util.Date` object or string representing the number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT (in essence, the value from `java.util.Date.getTime()` or `java.util.Calendar.getTime()`).
Time	Time	A 4-digit integer to represent time of the day in HHMM format. First 2-digit is the hour in 24-hour format. Last 2-digit is minutes.

64.1.3.3 Business Terms

Business terms are rule terms defined and managed by the system administrator through Oracle Application Server 11g Enterprise Manager. For more information on adding, defining, and deleting business terms, refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*. A business term consists of a key, a data type, an optional description, and an optional List of Values (LOV).

Table 64–3 lists the pre-defined business terms supported by User Messaging Preferences.

Table 64–3 Pre-defined Business Terms for User Messaging Preferences

Business Term	Data Type
Service Name	String
Process Name	String
System Code	String
Error Code	String
Occurrence Count	Number (Decimal)
Organization	String
Time	Number (Decimal)
Priority	String
Application	String
Application Type	String
Expiration Date	Date
From	String
To	String
Customer Name	String
Customer Type	String
Status	String
Amount	Number (Decimal)
Due Date	Date
Process Type	String
Expense Type	String
Total Cost	Number (Decimal)
Processing Time	Number (Decimal)
Order Type	String
Service Request Type	String
Group Name	String
Source	String
Classification	String
Duration	Number (Decimal)
User	String
Role	String

64.1.4 Rule Actions

For a given rule, a User Messaging Preferences user can define one of the following actions:

- **Broadcast to All**: send a broadcast message to all channels in the broadcast address list.
- **Failover**: Send a message serially to channels in the address list until one successful message is sent. This means performing a send to the next channel

when the current channel returns a failure status. User Messaging Preferences does not allow a user to specify a channel-specific status code or expiration time.

- **Do not send to Any Channel**: Do not send a message to any channel.

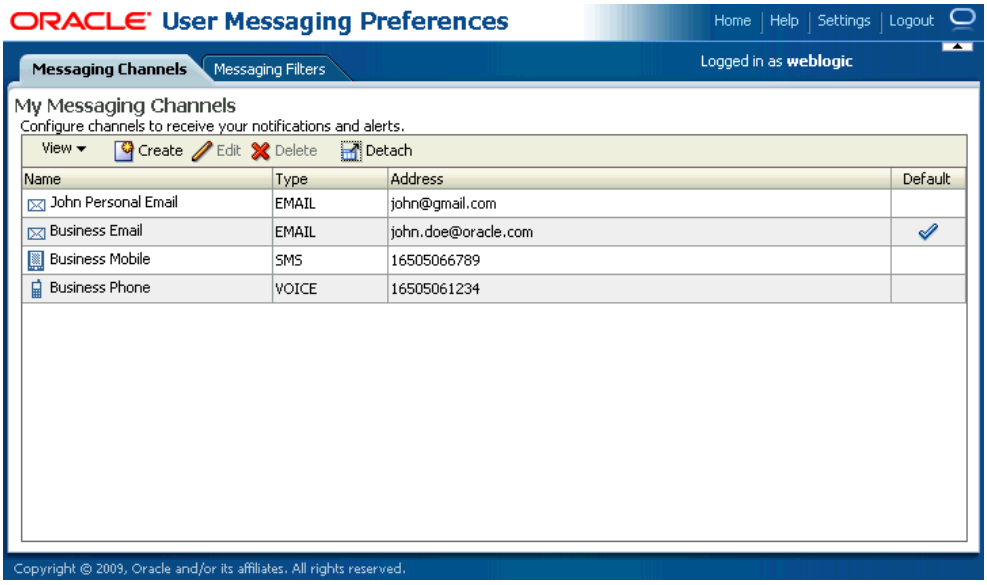
  Tip: User Messaging Preferences does not provide a filter action that instructs "do not send to a specified channel." A best practice is to specify only positive actions, and not negative actions in rules.

- **Default address**: if no action is defined, a message is sent to a default address or addresses, as defined in the Messaging Channels page in Enterprise Manager.

64.2 How to Manage Messaging Channels

Any channel that a user creates is associated with that user's system ID. In Oracle User Messaging Service, channels represent both physical channels, such as mobile phones, and also email client applications running on desktops, and are configurable on the The *Messaging Channels* tab (Figure 64–1).

Figure 64–1 Messaging Channels Tab



The *Messaging Channels* tab enables users to perform the following tasks:

64.2.1 Creating a Channel

To create a channel:

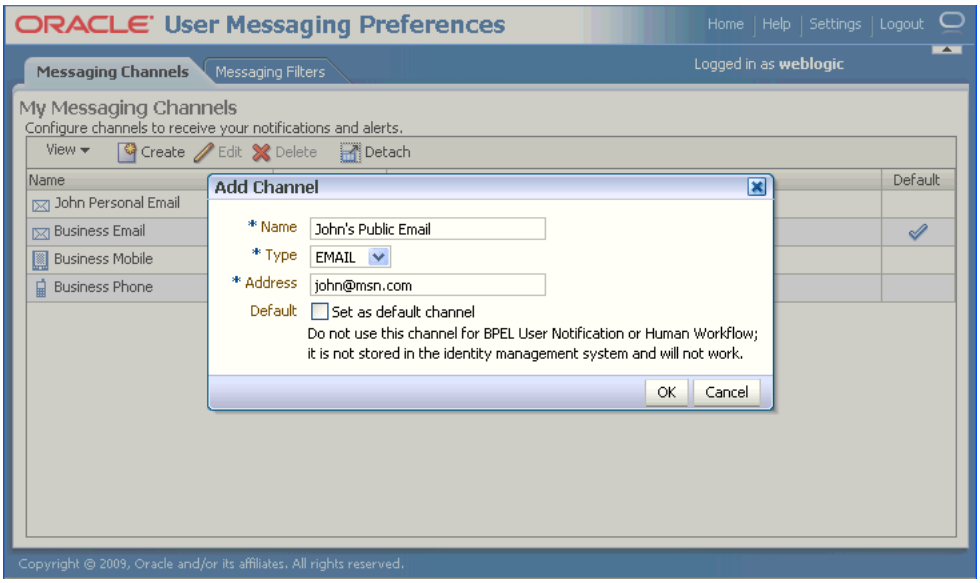
1. Click **Create** (Figure 64–2).

Figure 64–2 The Create Icon

2. Enter a name for the channel in the **Name** field (Figure 64–3).
3. Select the channel's transport type from the **Type** dropdown menu.
4. Enter the number or address appropriate to the transport type you selected.

5. Select the **Default** checkbox to set the channel as a default channel. You can have multiple default channels.

Figure 64–3 Creating a Channel

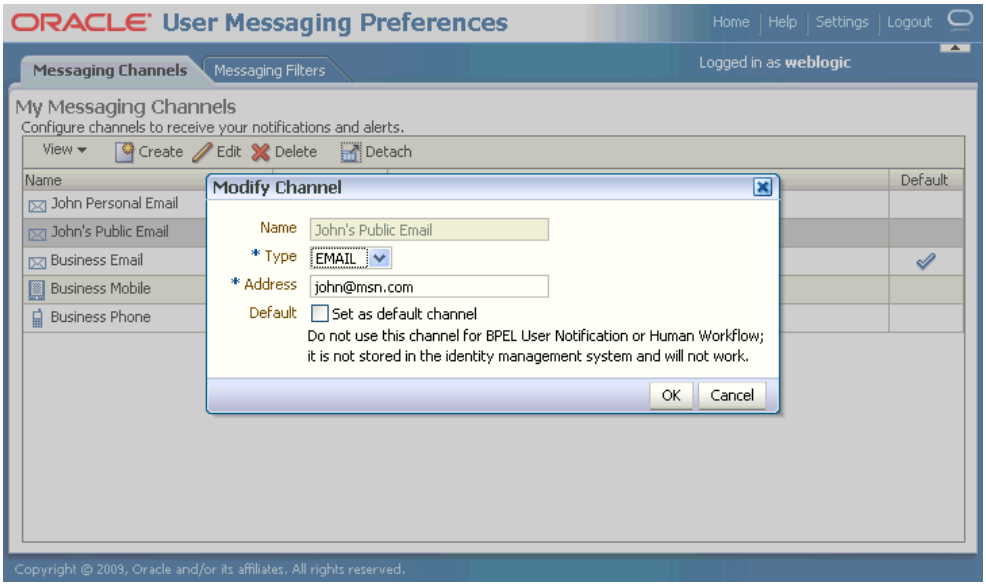


6. Click **OK** to create the channel. The channel appears on the *Channels* page. The *Channels* page enables you to edit or delete the channel.

64.2.2 Editing a Channel

To edit a channel, select it from the Channels list and click **Edit** (Figure 64–4). The editing page appears for the channel, which enables you to change the channel properties described in Section 64.2.1, "Creating a Channel".

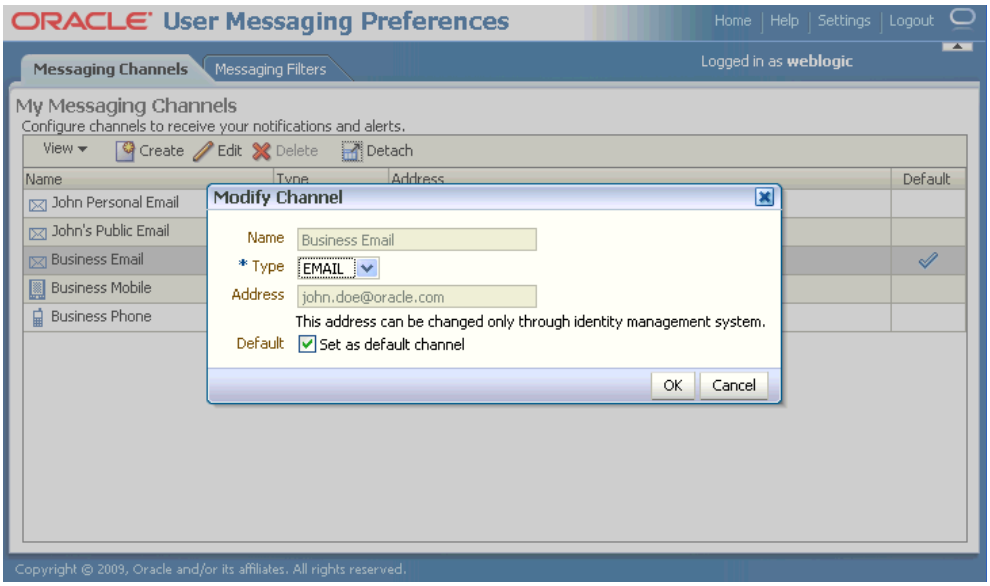
Figure 64–4 Edit a Channel



Certain channels are based on information retrieved from your user profile in the identity store, and this address cannot be modified by User Messaging Preferences

(Figure 64–5). The only operation that can be performed on such as channel is to make it the default.

Figure 64–5 Edit a Identity Store-Backed Channel



64.2.3 Deleting a Channel

To delete a channel, select it and click **Delete** (Figure 64–6).

Figure 64–6 The Delete Icon

64.2.4 Setting a Default Channel

You can configure one or more channels as default channels. Email is preconfigured as a default for receiving notifications. You can add or remove a channel as a default channel.

To set an additional channel as a default, select it, click **Edit**, and then click **Set as default channel**. A checkmark (Figure 64–7) appears next to the selected channel, designating it as a default means of receiving notifications. Repeat this procedure to add additional default channels, if required.

Figure 64–7 The Default Icon

64.3 Creating Contact Rules using Filters

The **Messaging Filters** tab (Figure 64–8) enables users to build filters that specify not only the type of notifications they want to receive, but also the channel through which to receive these notifications through a combination of comparison operators (such as *is equal to*, *is not equal to*), business terms that describe the notification type, content or source, and finally, the notification actions, which send the notifications to all channels, block channels from receiving notifications, or send notifications to the first available channel.

Figure 64–8 Messaging Filters Tab

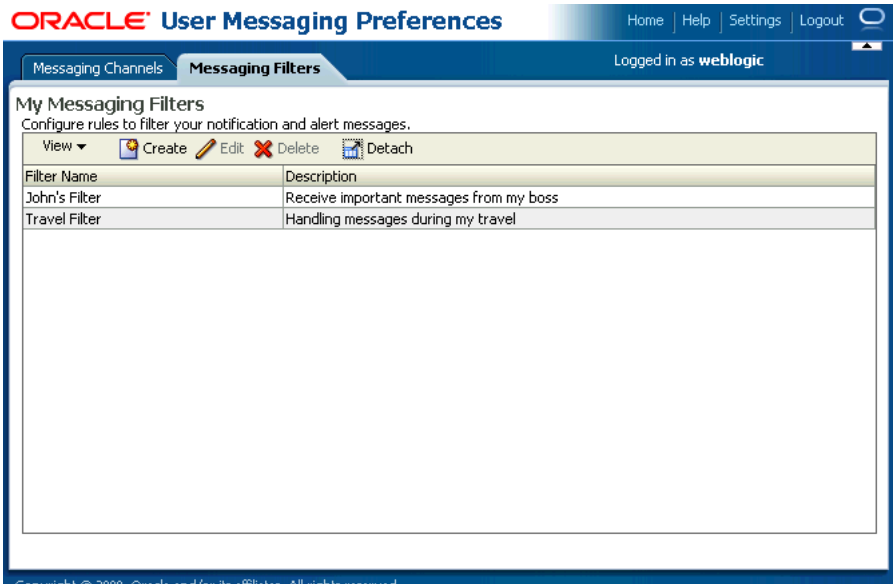
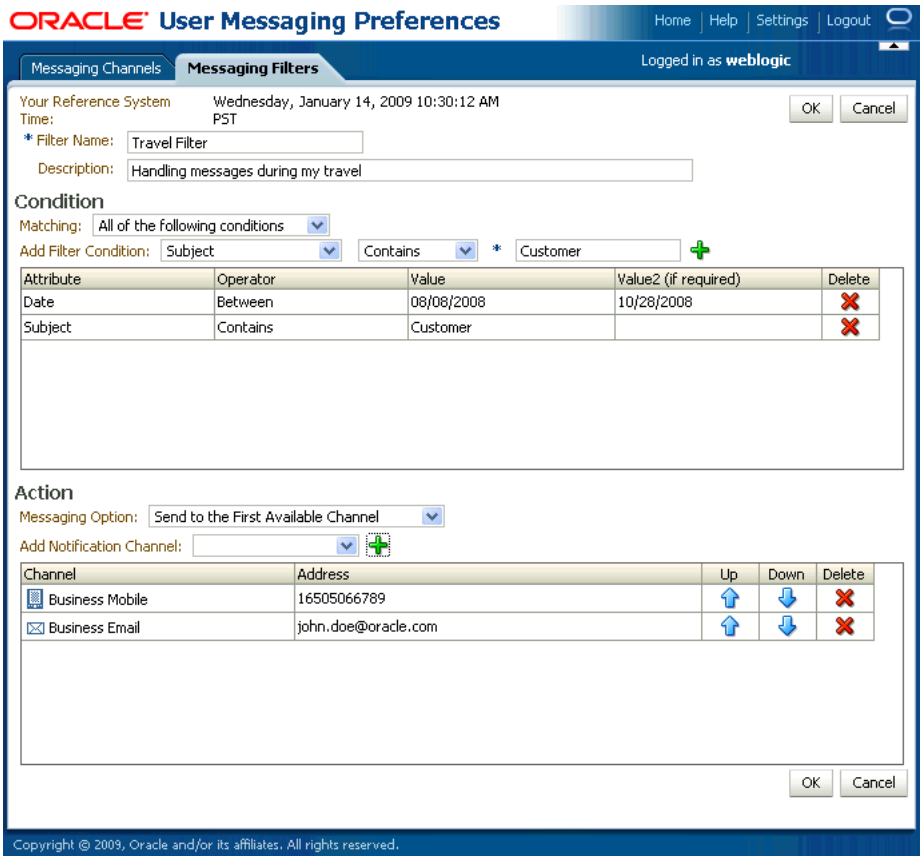


Figure 64–9 illustrates the creation of a filter called Travel Filter, by a user named weblogic, for handling notifications regarding Customers during his travel. Notifications that match all of the filter conditions are first directed to his "Business Mobile" channel. Should this channel become unavailable, Oracle User Messaging Service transmits the notifications as e-mails since the next available channel selected is Business Email.

Figure 64–9 Creating a Filter



64.3.1 Creating Filters

To create a filter:

1. Click **Create** (Figure 64–2). The Create Filter page appears (Figure 64–9).
2. Enter a name for the filter in the **Filter Name** field.
3. If needed, enter a description of the filter in the **Description** field.
4. Select whether notifications must meet all of the conditions or any of the conditions by selecting either the **All of the following conditions** or **Any of the following conditions** options.
5. Define the filter conditions using the lists and fields of the Condition section as follows:
   a. Select the notification's attributes. Refer to Table 64–3 for a list of these attributes.
   b. Combine the selected condition type with one of the comparison operators described in Table 64–1.
   c. Add appropriate values describing the attributes or operators.

   For instance, if you select the Date attribute, select one of the comparison operators and then select the appropriate dates from the date chooser.
6. Click **Add** (Figure 64–6) to add the attribute and the comparison operators to the table.

7. Repeat these steps to add more filter conditions. To delete a filter condition, click **Delete** (Figure 64–6).
8. Select one of the following delivery rules:
   - **Send Messages to all Selected Channels** -- Select this option to send messages to every listed channel.
   - **Send to the First Available Channel (Failover in the order)** -- Select this option to send messages matching the filter criteria to a preferred channel (set using the up and down arrows) or to the next available channel.
   - **Send No Messages** -- Select this option to block the receipt of any messages that meet the filter conditions.
9. To set the delivery channels, select a channel from the **Add Notification Channel** list and then click **Add** (Figure 64–6). To delete a channel, click **Delete** (Figure 64–6).
10. If needed, use the up and down arrows to prioritize channels. If available, the top-most channel receives messages meeting the filter criteria if you select **Send to the First Available Channel**.
11. Click **OK** to create the filter. Clicking **Cancel** discards the filter.

64.3.2 Editing a Filter

To edit a filter, first select it and then click **Edit** (Figure 64–9). The editing page appears for the filter, which enables you to add or change the filter properties described in Section 64.3.1, "Creating Filters".

64.3.3 Deleting a Filter

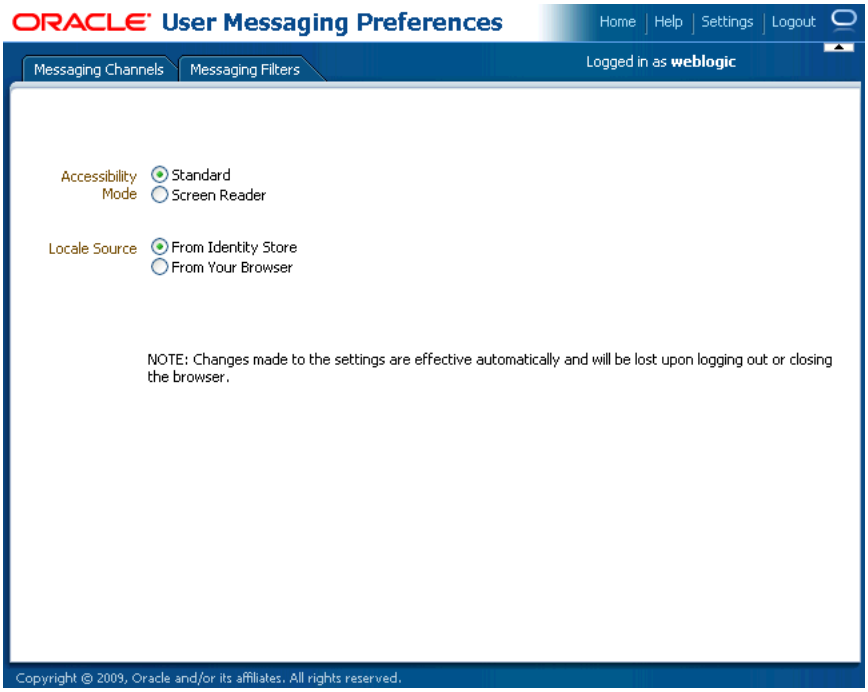
To delete a filter, first select it and then click **Delete** (Figure 64–6).

64.4 Configuring Settings

The **Settings** tab (Figure 64–10), accessed from the upper right area, enables users to set the following parameters:

- **Accessibility Mode**: select **Standard** or **Screen Reader**.
- **Locale Source**: select **From Identity Store** or **From Your Browser**.

Figure 64–10 Configuring Settings



Part XII

Appendices

This part describes Oracle SOA Suite appendixes.

This part contains the following appendixes:

A

BPEL Process Activities and Services

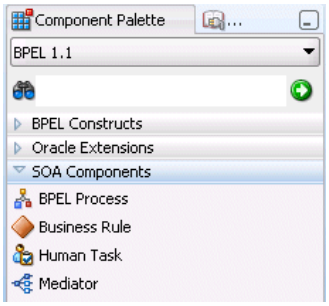
This appendix describes the activities and services that you use when designing a BPEL process in a SOA composite application.

This appendix includes the following sections:

A.1 Introduction to Activities and Components

When you expand **SOA Components** in the Component Palette of Oracle BPEL Designer, service components are displayed. Figure A–1 shows the components that display for a BPEL 1.1 process. A BPEL 2.0 process also shows the same components.

Figure A–1 SOA Components



See the following sections for additional details.

- BPEL process

  See Part II, "Using the BPEL Process Service Component"

- Business rule

  See Part IV, "Using the Business Rules Service Component"

- Human task

Part V, "Using the Human Workflow Service Component"

- Mediator

  See Part III, "Using the Oracle Mediator Service Component"

A.2 Introduction to BPEL 1.1 and 2.0 Activities

This section provides a brief overview of BPEL activities and provides references to other documentation that describes how to use these activities.

Oracle BPEL Designer includes BPEL 1.1 and BPEL 2.0 activities that are available for adding in a BPEL process. These activities enable you to perform specific tasks within a process. Some activities are available in both BPEL 1.1 and BPEL 2.0. Others are available in only BPEL 1.1 or BPEL 2.0.

To access these activities, go to the Component Palette of Oracle BPEL Designer. The activities display under either of two categories:

- **BPEL Constructs**: Displays core activities (also known as constructs) provided by standard BPEL 1.1 and 2.0 functionality. The activities in this category are displayed under additional subcategories of **Web Service**, **Activities**, and **Structured Activities** in BPEL 1.1 and **Web Service**, **Basic Activities**, and **Structured Activities** in BPEL 2.0.
- **Oracle Extensions**: Displays extension activities that add value and ease of use to BPEL 1.1 and 2.0 functionality

Table A–1 lists the available activities.

Table A–1 BPEL 1.1 and 2.0 Constructions and Extensions

Activity	Display Under...	Supported in BPEL 1.1	Supported in BPEL 2.0	For More Information
Assign	BPEL Constructs	Yes	Yes	Section A.2.2, "Assign Activity"
Assert	Oracle Extensions	Yes	No	Section A.2.3, "Assert Activity"
Bind Entity	Oracle Extensions	Yes	No	Section A.2.4, "Bind Entity Activity"
Compensate	BPEL Constructs	Yes	Yes	Section A.2.5, "Compensate Activity"
CompensateScope	BPEL Constructs	No	Yes	Section A.2.6, "CompensateScope Activity"
Create Entity	Oracle Extensions	Yes	No	Section A.2.7, "Create Entity Activity"
Dehydrate	Oracle Extensions	Yes	Yes	Section A.2.8, "Dehydrate Activity"
Email	Oracle Extensions	Yes	Yes	Section A.2.9, "Email Activity"
Empty	BPEL Constructs	Yes	Yes	Section A.2.10, "Empty Activity"
Exit	BPEL Constructs	No	Yes **Note:** Replaces the terminate activity in BPEL 2.0.	Section A.2.11, "Exit Activity"

Table A–1 (Cont.) BPEL 1.1 and 2.0 Constructions and Extensions

Activity	Display Under...	Supported in BPEL 1.1	Supported in BPEL 2.0	For More Information
Flow	BPEL Constructs	Yes	Yes	Section A.2.12, "Flow Activity"
FlowN	Oracle Extensions	Yes	No **Note**: Replaced by the forEach activity in BPEL 2.0	Section A.2.13, "FlowN Activity"
forEach	BPEL Constructs	No	Yes **Note:** Replaces the FlowN activity in BPEL 2.0.	Section A.2.14, "forEach Activity"
If	BPEL Constructs	No	Yes **Note**: Replaces the switch activity in BPEL 2.0.	Section A.2.15, "If Activity"
IM	Oracle Extensions	Yes	Yes	Section A.2.16, "IM Activity"
Invoke	BPEL Constructs	Yes	Yes	Section A.2.17, "Invoke Activity"
Java Embedding	Oracle Extensions	Yes	Yes	Section A.2.18, "Java Embedding Activity"
Partner Link	BPEL Constructs	Yes	Yes	Section A.2.19, "Partner Link Activity"
Phase	Oracle Extensions	Yes	Yes	Section A.2.20, "Phase Activity"
Pick	BPEL Constructs	Yes	Yes	Section A.2.21, "Pick Activity"
Receive	BPEL Constructs	Yes	Yes	Section A.2.22, "Receive Activity"
Receive Signal	Oracle Extensions	Yes	Yes	Section A.2.23, "Receive Signal Activity"
Remove Entity	Oracle Extensions	Yes	No	Section A.2.24, "Remove Entity Activity"
RepeatUntil	BPEL Constructs	No	Yes	Section A.2.25, "RepeatUntil Activity"
Replay	Oracle Extensions	Yes	Yes	Section A.2.26, "Replay Activity"
Reply	BPEL Constructs	Yes	Yes	Section A.2.27, "Reply Activity"
Rethrow	BPEL Constructs	No	Yes	Section A.2.28, "Rethrow Activity"
Scope	BPEL Constructs	Yes	Yes	Section A.2.29, "Scope Activity"
Sequence	BPEL Constructs	Yes	Yes	Section A.2.30, "Sequence Activity"
Signal	Oracle Extensions	Yes	Yes	Section A.2.31, "Signal Activity"

Table A–1 (Cont.) BPEL 1.1 and 2.0 Constructions and Extensions

Activity	Display Under...	Supported in BPEL 1.1	Supported in BPEL 2.0	For More Information
SMS	Oracle Extensions	Yes	Yes	Section A.2.32, "SMS Activity"
Switch	BPEL Constructs	Yes	No **Note**: Replaced by the if activity in BPEL 2.0.	Section A.2.33, "Switch Activity"
Terminate	BPEL Constructs	Yes	No **Note**: Replaced by the exit activity in BPEL 2.0	Section A.2.34, "Terminate Activity"
Throw	BPEL Constructs	Yes	Yes	Section A.2.35, "Throw Activity"
Transform	Oracle Extensions	Yes	Yes	Section A.2.36, "Transform Activity"
User Notification	Oracle Extensions	Yes	Yes	Section A.2.37, "User Notification Activity"
Validate	Oracle Extensions (in BPEL 1.1) BPEL Constructs (in BPEL 2.0)	Yes	Yes	Section A.2.38, "Validate Activity"
Voice	Oracle Extensions	Yes	Yes	Section A.2.39, "Voice Activity"
Wait	BPEL Constructs	Yes	Yes	Section A.2.40, "Wait Activity"
While	BPEL Constructs	Yes	Yes	Section A.2.41, "While Activity"

For more information about activities, see the *Business Process Execution Language for Web Services Specification Version 1.1* or the *Web Services Business Process Execution Language Specification Version 2.0* by visiting the following URL:

`http://www.oasis-open.org`

A.2.1 Tabs Common to Many Activities

While each activity performs specific tasks, many activities include tabs that enable you to perform similar tasks. This section describes these common tabs.

A.2.1.1 Annotations Tab

The **Annotations** tab displays on all activities and enables you to provide descriptions in activities in the form of code comments and name and pair value assignments.

Note that the **Annotations** tab does not provide a method for changing the order of annotations. As a work around, change the order of annotations in the **Source** view of the project's BPEL file in Oracle BPEL Designer.

A.2.1.2 Assertions Tab

The **Assertions** tab displays in invoke, receive, reply, and the onMessage branches of pick and scope activities. A set of assertions are executed upon receipt of a callback message at a request-response operation in these activities. The assertions specify an

XPath expression that, when evaluated to false, causes a BPEL fault to be thrown from the activity. This provides an alternative to using a potentially large number of switch, assign, and throw activities after a partner callback.

Note: This tab is only available in BPEL 1.1 projects.

For more information, see the online help for this tab and Section 11.14, "Throwing Faults with Assertion Conditions."

A.2.1.3 Correlations Tab

The **Correlations** tab displays in invoke, receive, and reply activities, the onMessage branch of pick activities, and the OnMessage branch of scope activities. Correlation sets address complex interactions between a process and its partners by providing a method for explicitly specifying correlated groups of operations within a service instance. A set of correlation tokens is defined as a set of properties shared by all messages in the correlated group.

For more information, see the online help for this tab and Section 8.5, "Using Correlation Sets in an Asynchronous Service."

A.2.1.4 Documentation Tab

The **Documentation** tab enables you to embed human documentation in the activities of a BPEL file. These comments only display in the source code of the BPEL file. Example A–1 provides details.

Example A–1 Documentation Tab

```
<invoke>
. . .
   <documentation>
      Invokes the credit rating service partner link
   </documentation>
. . .
```

Note: This tab is only available in BPEL 2.0 projects.

A.2.1.5 Headers Tab

The **Headers** tab displays in invoke, receive, and reply activities, and the onMessage branch of pick and scope (for BPEL 1.1) activities. You create header variables for use with the Advanced Queuing (AQ), File, FTP, MQ, and Java Message Service (JMS) adapters.

For more information, see the online help for this tab and *Oracle Fusion Middleware User's Guide for Technology Adapters*

A.2.1.6 Properties Tab

The **Properties** tab displays in invoke, receive, and reply activities, and the onMessage branch of pick and scope activities.You can define normalized message header properties for Oracle BPEL Process Manager, Oracle Mediator, Oracle JCA adapters, and Oracle B2B.

For more information, see the online help for this tab and Appendix H, "Normalized Message Properties."

A.2.1.7 Skip Condition Tab

The **Skip Condition** tab displays in most activities and enables you to specify an XPath expression that, when evaluated to true, causes the activity to be skipped. This extension provides an alternative to the case pattern of a switch activity that you use to make an activity conditional.

Note: This tab is only available in BPEL 1.1 projects.

For more information, see the online help for this tab and Section 10.5, "Specifying XPath Expressions to Bypass Activity Execution."

A.2.1.8 Source and Targets Tabs

The **Sources** and **Targets** tabs enable you to define the source and target activities to execute in a flow activity. This feature enables you to synchronize the execution of activities within a flow activity to ensure that a target activity only executes after a source activity have completed.

For more information, see the online help for this tab and Section 9.2.3, "Synchronizing the Execution of Activities in a Flow Activity."

A.2.1.9 Timeout Tab

The **Timeout** tab displays in receive activities and provides a timeout setting for request-response operations. This provides an alternative to the onMessage and onAlarm branches of a pick activity that you must use when you want to specify a time out duration for partner callbacks.

Note: This tab is only available in BPEL 1.1 projects.

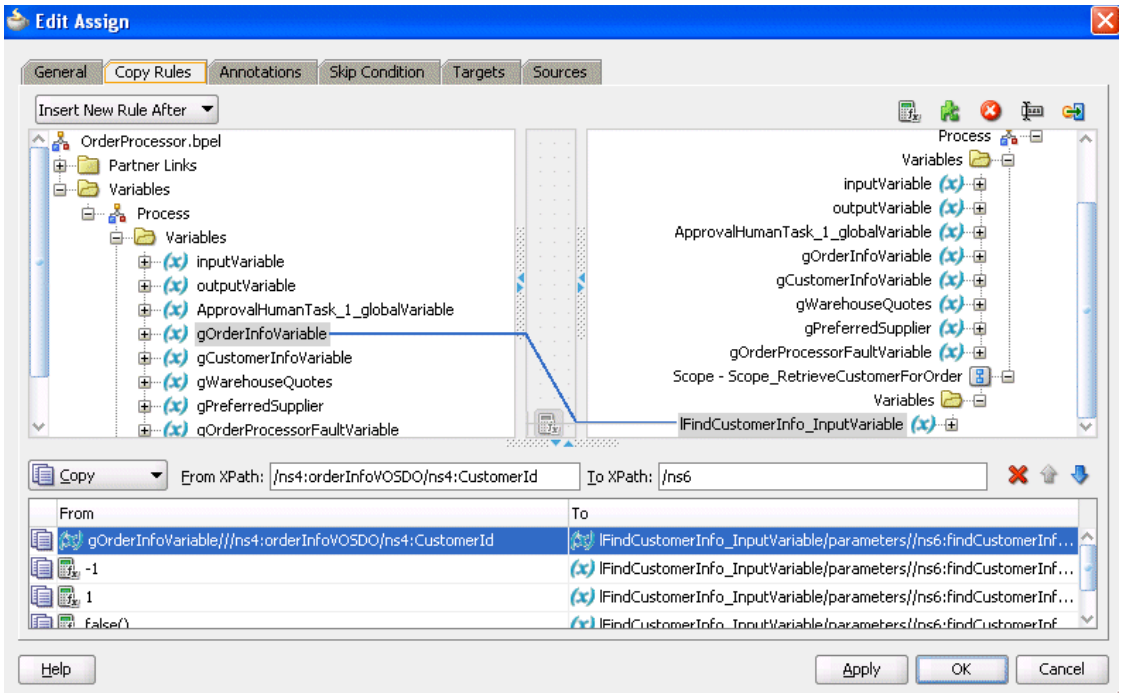
For more information, see the online help for this tab and Section 14.3, "Setting Timeouts for Request-Response Operations in Receive Activities."

A.2.2 Assign Activity

This activity provides a method for data manipulation, such as copying the contents of one variable to another. Copy operations enable you to transfer information between variables, expressions, endpoints, and other elements.

Figure A–2 shows the **Copy Rules** tab of the Assign dialog for BPEL 1.1. You drag the source node to the target node to create a BPEL copy rule from the source to the target node. This action creates a line that connects the source and target types. The copy rule is displayed in the **From** and **To** sections at the bottom of the dialog.

Figure A–2 Copy Rules Tab of Assign Activity Dialog



The **Select Insertion Mode** list above the source node section enables you to insert the next copy rule you create either after or before the rule selected at the bottom of the dialog.

Icons display above the target node that enable you to perform the following tasks (from left to right) on target nodes.

- **Expression** icon: Drag this icon to a target node to invoke the Expression Builder dialog for assigning an XPath expression to that node.
- **Literal** (BPEL 2.0 specification) icon or **XML Fragment** (BPEL 1.1 specification) icon: Drag this icon to a target node to invoke a dialog for assigning a literal (if the BPEL project supports the BPEL 2.0 specification) or XML fragment (if the BPEL project supports the BPEL 1.1 specification) to that target node.
- **Remove** icon: Drag this icon to a target node to create a `bpelx:remove` extension rule.
- **Rename** icon: Drag this icon to rename a target node. This adds a `bpelx:rename` extension rule with an `elementTo` attribute.
- **Recast** icon: Drag this icon to recast a target node. This adds a `bpelx:rename` extension rule with a `typeCastTo` attribute. This results in an `xsi:type` attribute in the XML output.

You can also change a selected copy rule to a `bpelx` extension type (`bpelx:copyList`, `bpelx:insertAfter`, `bpelx:insertBefore`, or `bpelx:append`).

The method of selection differs between BPEL 1.1 and BPEL 2.0.

Figure A–3 shows how you select an extension type in BPEL 1.1. You select a copy rule, select the **Copy** dropdown list, and then select the appropriate extension.

Figure A–3 Copy Rule Converted to bpelx Extension in BPEL 1.1

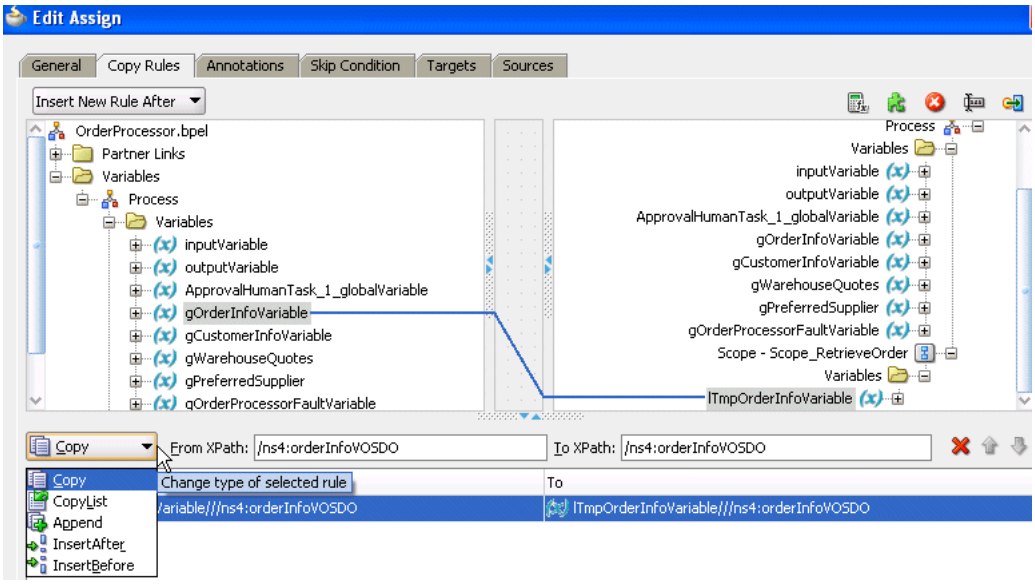
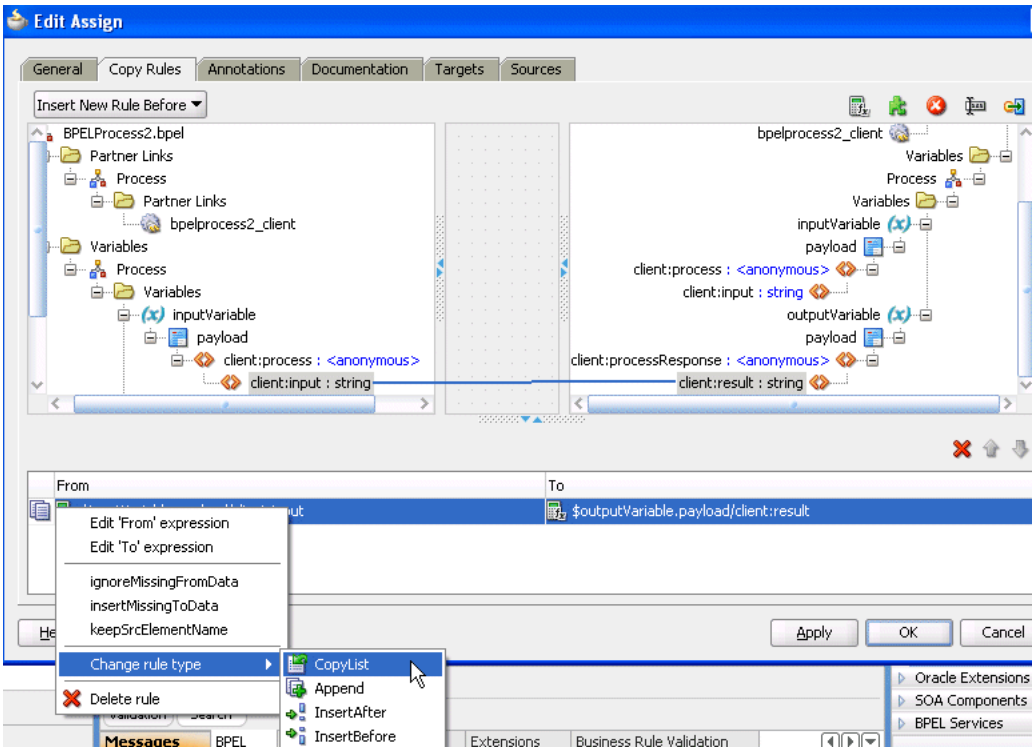


Figure A–4 shows how you select an extension type in BPEL 2.0. You right-click a copy rule, select **Change rule type**, and then select the appropriate extension.

Figure A–4 Copy Rule Converted to bpelx Extension in BPEL 2.0



For more information about manipulating XML data with `bpelx` extensions, see Section 6.14, "Manipulating XML Data with bpelx Extensions."

In the **From** and **To** XPath fields, you can also place your cursor over the icon to the left of the source type to display the operation being performed (for example, copy,

append, and so on). Each operation type is represented by a different icon. You can also right-click a copy rule to display a list of actions to perform:

- **Edit 'From' Expression** or **Edit 'To' Expression**: Select this option to edit XPath expression values when the created copy rule contains a query for the source or target node. This selection invokes the Expression Builder dialog. The menu option that displays is based on the current content of your copy rule selection.
- **ignoreMissingFromData**: Select this option to toggle the `ignoreMissingFromData` attribute on the copy rule on and off. When toggled on, this suppresses any `bpel:selectionFailure` standard faults.
- **insertMissingToData**: Select this option to toggle the `insertMissingToData` attribute on the copy rule on and off.
- **keepSrcElementName** (in BPEL 2.0 projects only): Select this option to toggle the `keepSrcElementName` attribute on the copy rule on and off. This option enables you to replace the element name of the destination (as selected by the `to-spec`) with the element name of the source.
- **Change Rule Type** (in BPEL 2.0 projects only): Select this option to change the type of the selected rule to one of the BPEL extension rules: `bpelx:copyList`, `bpelx:insertAfter`, `bpelx:insertBefore`, or `bpelx:append`.
- **Delete rule**: Select this option to delete the selected rule.

For more information about the ignoreMissingFromData, insertMissingToData, and keepSrcElementName attributes, see Section 6.14.7, "How to Use Assign Extension Attributes."

The icons above the **To** section enable you to delete, move up, and move down a selected copy rule.

For more information about the assign activity, see the online Help for the Copy Rules dialog and Chapter 6, "Manipulating XML Data in a BPEL Process."

Note: If an assign activity contains multiple `bpelx:append` settings, it must be split into two assign activities. Otherwise, `bpelx:append` is moved to the end of the list each time, which can cause problems. As a work around, move it manually.

A.2.3 Assert Activity

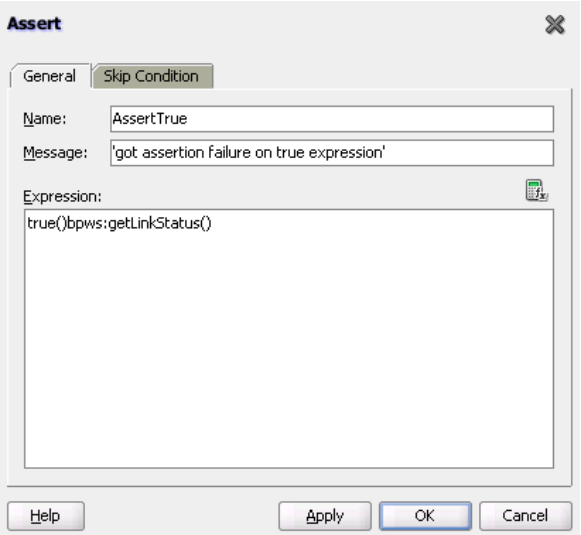
This activity enables you to perform an assertion on a specified expression.

This is a standalone activity in which to specify assertions. You can also specify assertions from the **Assertions** tab in invoke activities, receive activities, and the onMessage branch of pick and scope activities.

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

Figure A–5 shows the Assert dialog.

Figure A–5 Assert Dialog



For more information about the standalone assert activity, see Section 11.14.7, "Assertion Conditions in a Standalone Assert Activity" and Section 11.14.10, "What Happens When You Create Assertion Conditions."

A.2.4 Bind Entity Activity

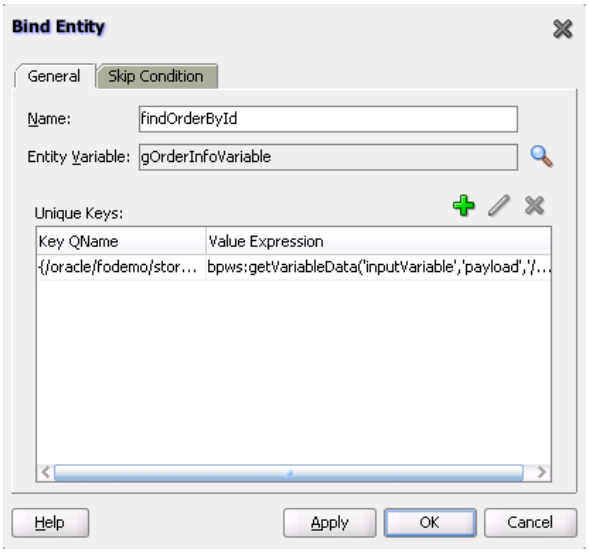
This activity enables you to select the entity variable to act as the data handle to access and plug in different data provider service technologies.

The entity variable can be used with an Oracle Application Development Framework (ADF) Business Component data provider service using service data object (SDO)-based data. The entity variable enables you to specify BPEL data operations to be performed by an underlying data provider service. The data provider service performs the data operations in a data store behind the scenes and without use of other data store-related features provided by Oracle BPEL Process Manager (for example, the database adapter). This action enhances Oracle BPEL Process Manager runtime performance and incorporates native features of the underlying data provider service during compilation and runtime.

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

Figure A–6 shows the Bind Entity dialog.

Figure A–6 Bind Entity Dialog



A.2.5 Compensate Activity

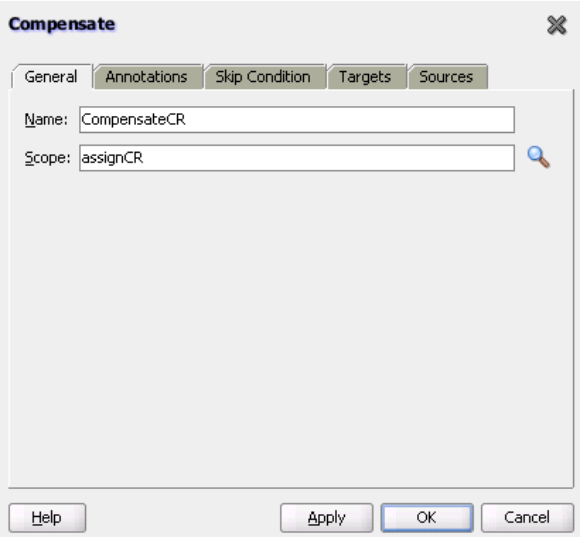
This activity invokes compensation on an inner scope activity that has successfully completed. This activity can be invoked only from within a fault handler or another compensation handler. Compensation occurs when a process cannot complete several operations after completing others. The process must return and undo the previously completed operations. For example, assume a process is designed to book a rental car, a hotel, and a flight. The process books the car and the hotel, but cannot book a flight for the correct day. In this case, the process performs compensation by unbooking the car and the hotel.

The compensation handler is invoked with the compensate activity, which names the scope on which the compensation handler is to be invoked.

Figure A–7 shows the Compensate dialog in BPEL 1.1. You can perform the following tasks:

- Click the **General** tab to provide the activity with a meaningful name.
- Select the **scope** activity on which the compensation handler is to be invoked.

Figure A–7 Compensate Dialog



In BPEL 2.0, the Compensate dialog does not include a **Skip Condition** tab.

For more information about the compensate activity, see Section 11.12, "Using Compensation After Undoing a Series of Operations."

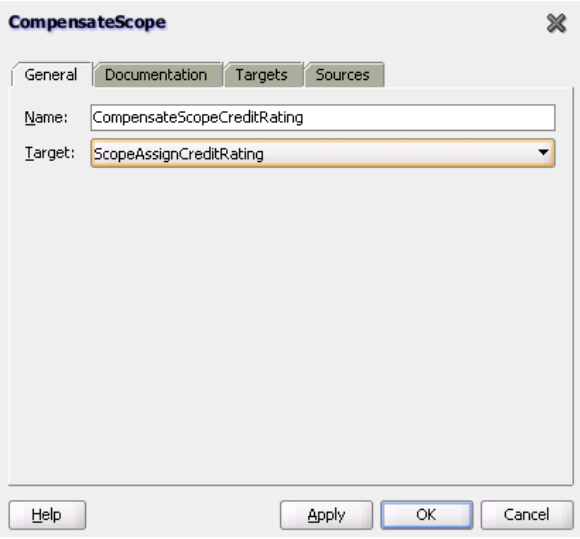
A.2.6 CompensateScope Activity

This activity enables you 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.

> **Note:** This activity is only supported in BPEL 2.0 projects.

Figure A–8 shows the CompensateScope dialog.

Figure A–8 CompensateScope Dialog



For more information about the compensateScope activity, see Section 11.12, "Using Compensation After Undoing a Series of Operations"

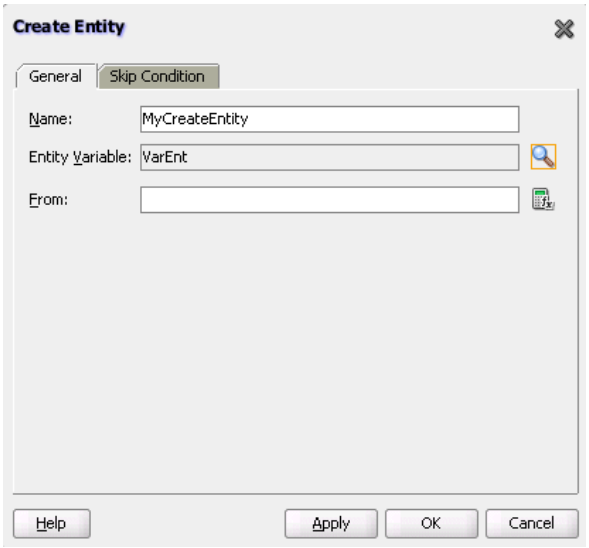
A.2.7 Create Entity Activity

This activity enables you to create an entity variable. The entity variable can be used with an Oracle ADF Business Component data provider service using SDO-based data.

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

Figure A–9 shows the Create Entity dialog.

Figure A–9 Create Entity Dialog



For more information, see Section 6.2, "Delegating XML Data Operations to Data Provider Services."

A.2.8 Dehydrate Activity

By default, dehydration points are set on activities such as a wait and a receive. The dehydrate activity enables you to explicitly specify a dehydration point. This activity acts as a dehydration point to automatically maintain long-running asynchronous processes and their current state information in a database while they wait for asynchronous callbacks. 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 reliability and scalability.

The `bpelx:dehydrate` extension implements dehydration. For BPEL projects that support BPEL version 1.1, the syntax is as follows:

```
<bpelx:dehydrate name="DehydrateInstance"/>
```

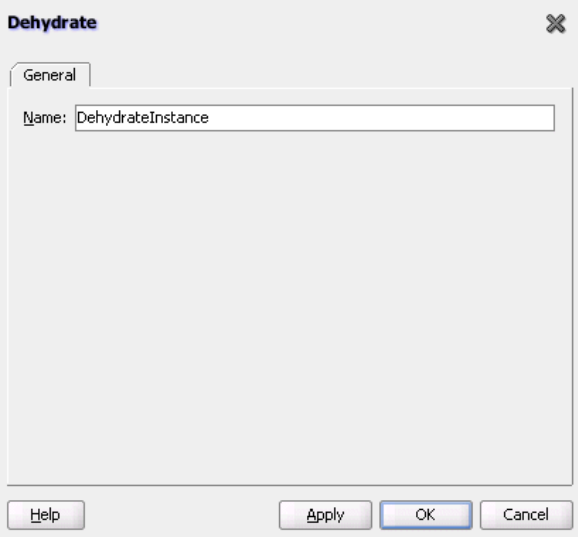
For BPEL projects that support BPEL version 2.0, the syntax is as shown in Example A–2.

Example A–2 bpelx:dehydrate Extension in BPEL 2.0

```
<extensionActivity>
      <bpelx:dehydrate name="DehydrateInstance"/>
 </extensionActivity>
```

Figure A–10 shows the Dehydrate dialog in BPEL 2.0.

Figure A–10 Dehydrate Dialog



In BPEL 1.1, the Dehydrate dialog includes a **Skip Condition** tab.

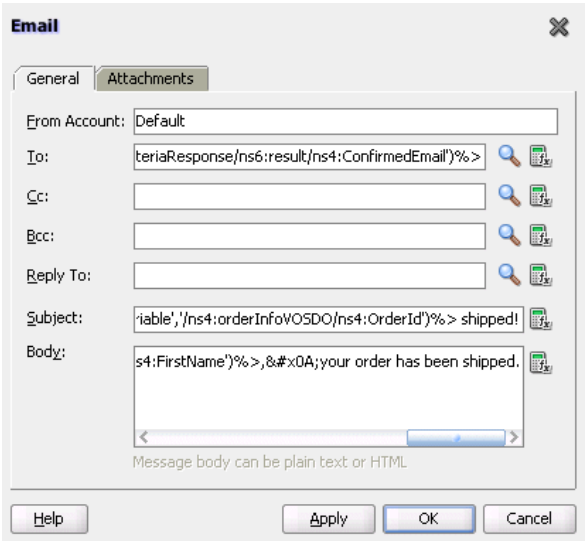
A.2.9 Email Activity

This activity enables you to send an email notification about an event.

For example, an online shopping business process of an online bookstore sends a courtesy email message to you after the items are shipped. The business process calls the notification service with your user ID and notification message. The notification service gets the email address from Oracle Internet Directory.

Figure A–11 shows the Email dialog in BPEL 1.1 and BPEL 2.0.

Figure A–11 Email Dialog



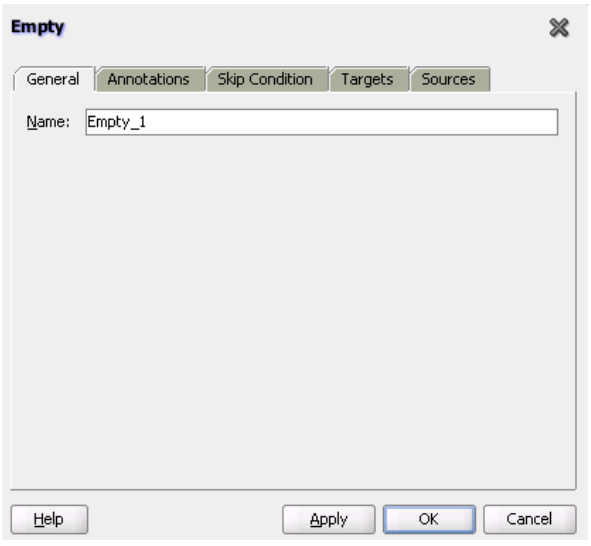
For more information about the email activity, see Section 16.3.1, "How To Configure the Email Notification Channel."

A.2.10 Empty Activity

This activity enables you to insert a no-operation instruction into a process. This activity is useful when you must use an activity that does nothing (for example, when a fault must be caught and suppressed).

Figure A–12 shows the Empty dialog in BPEL 1.1.

Figure A–12 Empty Dialog



In BPEL 2.0, the Empty dialog includes a **Documentation** tab and does not include a **Skip Condition** tab.

For more information about the empty activity, see Section 11.10.8, "How to Create an Empty Activity to Insert No-Op Instructions into a Business Process."

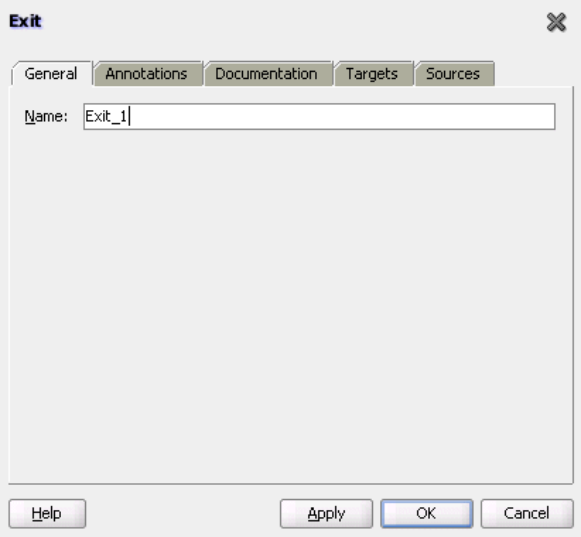
A.2.11 Exit Activity

This activity enables you to immediately end all currently running activities on all parallel branches without involving any termination handling, fault handling, or compensation handling mechanisms.

> **Note:** This activity replaces the terminate activity in BPEL 2.0 projects.

Figure A–13 shows the Exit dialog.

Figure A–13 Exit Dialog



For more information about the exit activity, see Section 11.13.2, "Immediately Ending a Business Process Instance with the Exit Activity in BPEL 2.0"

A.2.12 Flow Activity

This activity enables you to specify one or more activities to be performed concurrently. A flow activity completes when all activities in the flow have finished processing. Completion of a flow activity includes the possibility that it can be skipped if its enabling condition is false.

For example, assume you use a flow activity to enable two loan offer providers (United Loan service and Star Loan service) to start in parallel. In this case, the flow activity contains two parallel activities – the sequence to invoke the United Loan service and the sequence to invoke the Star Loan service. Each service can take an arbitrary amount of time to complete their loan processes.

Figure A–14 shows an initial flow activity with its two panels for parallel processing. You drag activities into both panels to create parallel processing. When complete, a flow activity looks like that shown in Figure A–15.

Figure A–14 Flow Dialog (At Time of Creation)

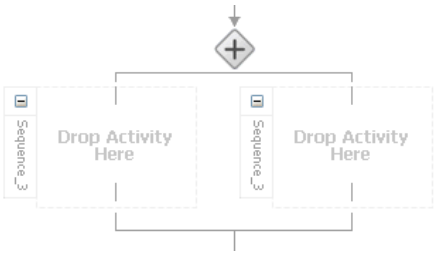
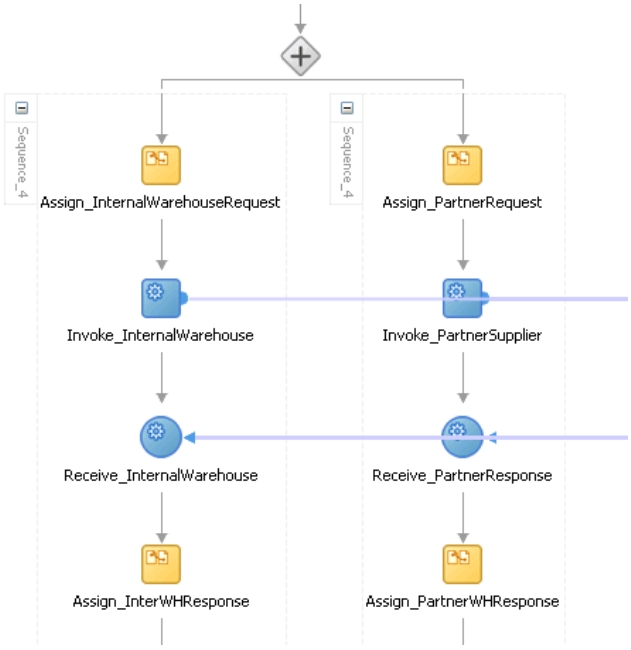


Figure A–15 Flow Dialog (After Design Completion)



You can also synchronize the execution of activities within a flow activity. This ensures that certain actives only execute after other activities have completed.

> **Note:** Oracle's BPEL implementation executes flows in the same, single execution thread of the BPEL process and not in separate threads.

For more information about the flow activity, see Section 9.2, "Creating a Parallel Flow."

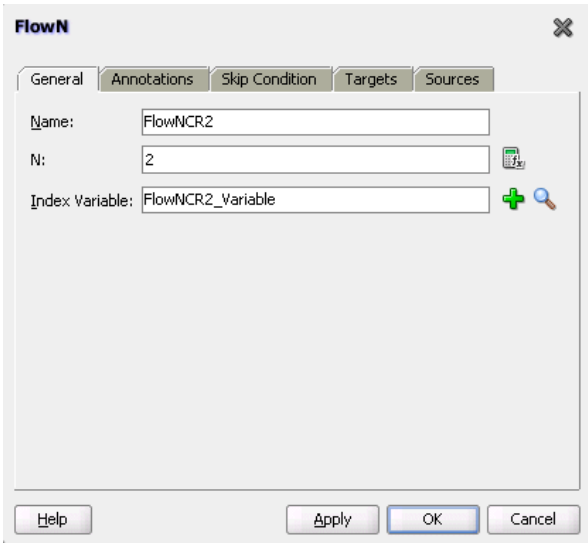
A.2.13 FlowN Activity

This activity enables you to create multiple flows equal to the value of `N`, which is defined at runtime based on the data available and logic within the process. An index variable increments each time a new branch is created, until the index variable reaches the value of `N`.

> **Note:** This activity is replaced by the forEach activity in BPEL 2.0 projects.

Figure A–16 shows the FlowN dialog.

Figure A–16 FlowN Dialog



For more information about the flowN activity, see Section 9.3.1, "Customizing the Number of Flow Activities with the flowN Activity in BPEL 1.1."

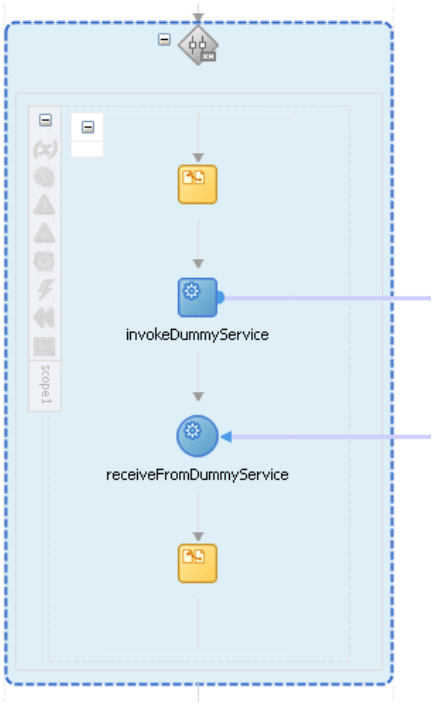
A.2.14 forEach Activity

This activity enables you to process multiple sets of activities sequentially or in parallel. The forEach activity executes its contained (child) scope activity exactly *N*+1 times, where *N* equals the final counter value minus the 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 use a scope activity.

Note: This activity replaces the flowN activity in BPEL 2.0 projects.

Figure A–17 shows a forEach activity with its contained scope.

Figure A–17 forEach Activity



For more information about the forEach activity, see Section 9.3.2, "Processing Multiple Sets of Activities with the forEach Activity in BPEL 2.0."

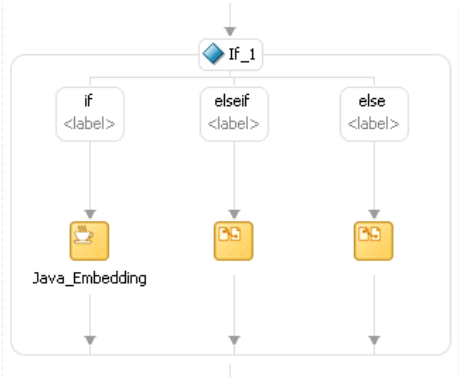
A.2.15 If Activity

This activity enables you to define conditional behavior for specific activities to decide between two or more branches. Only one activity is selected for execution from a set of branches.

Note: This activity replaces the switch activity in BPEL 2.0 projects.

Figure A–18 shows an if activity with the following defined if, elseif, and else branches.

Figure A–18 If Activity

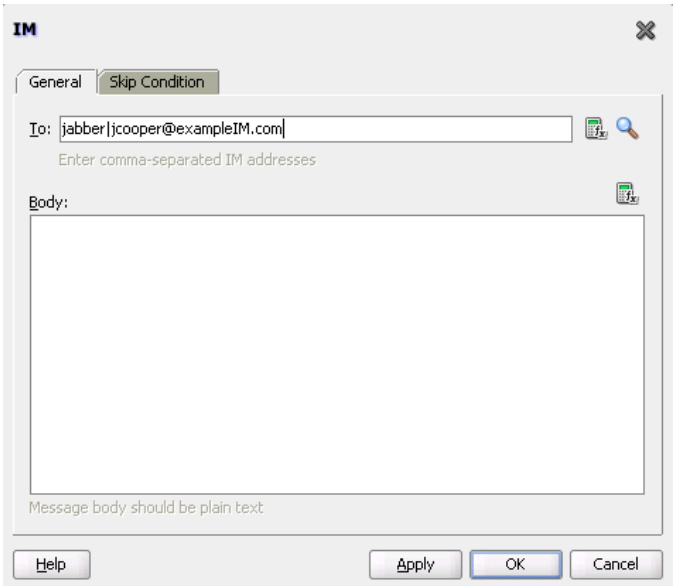


For more information about the if activity, see Section 10.2.2, "Defining Conditional Branching with the If Activity in BPEL 2.0."

A.2.16 IM Activity

This activity enables you to send an automatic, asynchronous instant message (IM) notification to a user, group, or destination address. Figure A–19 shows the IM dialog in BPEL 1.1.

Figure A–19 IM Dialog



In BPEL 2.0, the IM dialog does not include a **Skip Condition** tab.

For more information, see Section 16.3.2, "How to Configure the IM Notification Channel."

A.2.17 Invoke Activity

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

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

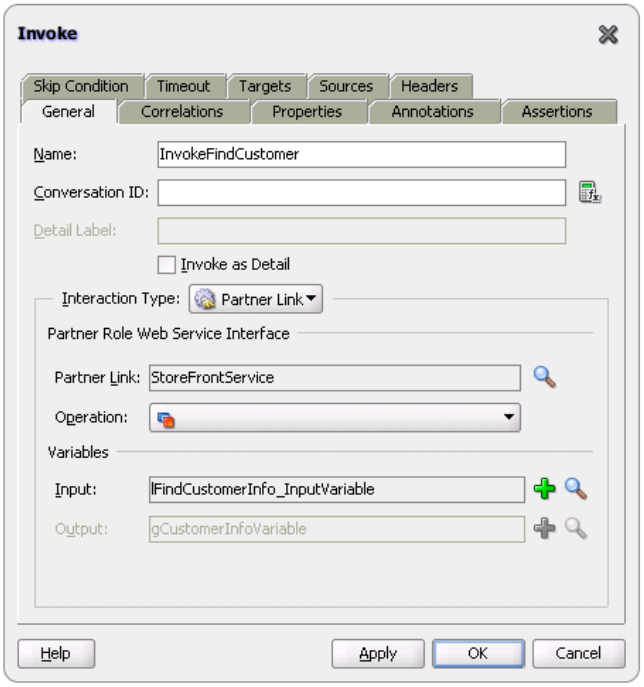
The invoke activity supports the `bpelx:inputProperty` and `bpelx:outputProperty` that facilitate the passing of properties through the SOAP header and the obtaining of SOA runtime system properties for useful information such as the **tracking.compositeInstanceId** and **tracking.conversationId**.

Figure A–20 shows the Invoke dialog in BPEL 1.1. You can perform the following tasks:

- Provide the activity with a meaningful name.
- Select the partner link for which to specify an operation.

- Select the operation to be performed.
- Automatically create a variable or select an existing variable in which to transport the data (payload).

Figure A–20 Invoke Dialog



In BPEL 2.0, the Invoke dialog does not include an **Assertions** tab, **Timeout** tab, or **Skip Condition** tab.

For more information about the invoke activity, see the following:

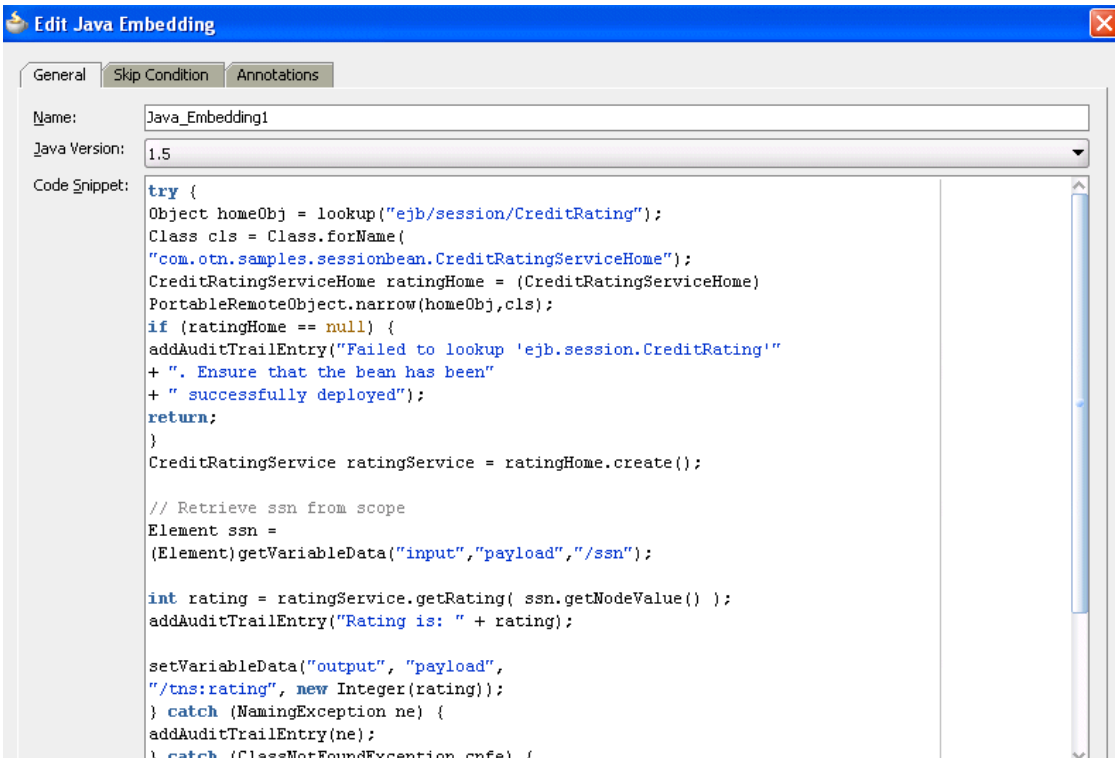
- Chapter 5, "Introduction to Interaction Patterns in a BPEL Process"
- Section 6.17, "Mapping WSDL Message Parts in BPEL 2.0"
- Section 7.2.2.3, "Invoke Activity for Performing a Request"
- Section 8.2.1.2, "Adding an Invoke Activity"
- Section 11.9.2, "How to Return a Fault in an Asynchronous Interaction"
- Section 11.14, "Throwing Faults with Assertion Conditions"

A.2.18 Java Embedding Activity

This activity enables you to add custom Java code to a BPEL process using the Java BPEL `exec` extension `bpelx:exec`. This is useful when you have Java code that can perform a function, and want to use this existing code instead of starting over. In BPEL 2.0 projects, the `bpelx:exec` extension and Java code are wrapped in an `<extensionActivity>` element.

Figure A–21 shows the Edit Java Embedding dialog in BPEL 1.1.

Figure A–21 Edit Java Embedding Dialog



In BPEL 2.0, the Invoke dialog does not include the **Skip Condition** tab.

For more information about the Java embedding activity, see Chapter 13, "Incorporating Java and Java EE Code in a BPEL Process."

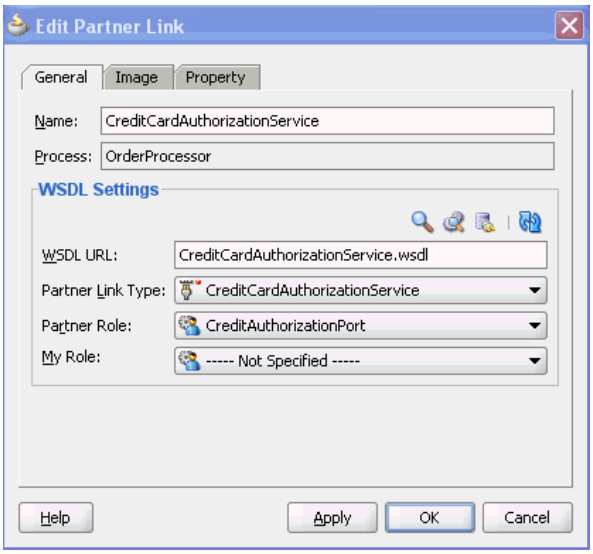
A.2.19 Partner Link Activity

This service enables you to define the external services with which your process interacts. A partner link type characterizes the conversational relationship between two services by defining the roles played by each service in the conversation and specifying the port type provided by each service to receive messages within the conversation. For example, if you are creating a process to interact with a Credit Rating Service and two loan provider services (United Loan and Star Loan), you create partner links for all three services.

Figure A–22 shows the Partner Link dialog in BPEL 1.1. You provide the following details:

- A meaningful name for the service.
- The web services description language (WSDL) file of the external service.
- The actual service type (defined as **Partner Link Type**).
- The role of the service (defined as **Partner Role**).
- The role of the process requesting the service (defined as **My Role**).

Figure A–22 Partner Link Activity



In BPEL 2.0, the Partner Link dialog also includes the **Documentation** tab.

For more information about partner links, see Chapter 8, "Invoking an Asynchronous Web Service from a BPEL Process."

A.2.20 Phase Activity

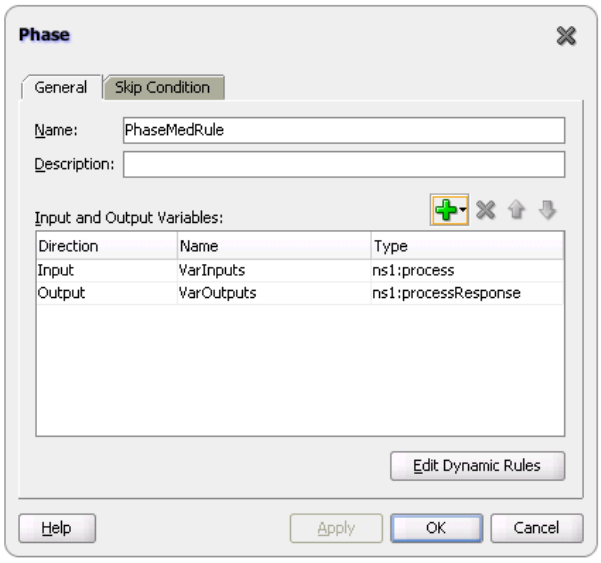
This activity creates Oracle Mediator and business rules service components for integration with a BPEL process. You create message request input and message response output variables and design business rules for evaluating variable content for the BPEL process.

When you complete these tasks, the following activities and service components are created:

- An assign activity that includes the message request input and message response output variables.
- An invoke activity, which is automatically designed to invoke an Oracle Mediator partner link in the BPEL process.
- The Oracle Mediator partner link, which is automatically designed to route the message request input variable to the business rules service component in the SOA composite application of which this BPEL process is a part. The business rules service component displays in the SOA Composite Editor. Oracle Mediator also displays as a service component in the SOA Composite Editor.
- The business rules service component, which evaluates the content of the message request input variable and returns the results in the message response output variable to Oracle Mediator. Oracle Mediator then makes a routing decision and routes the message to the correct target destinations.

Figure A–23 shows Phase dialog in BPEL 1.1.

Figure A–23 Phase Dialog



In BPEL 2.0, the Phase dialog includes the **Documentation** tab and does not include the **Skip Condition** tab.

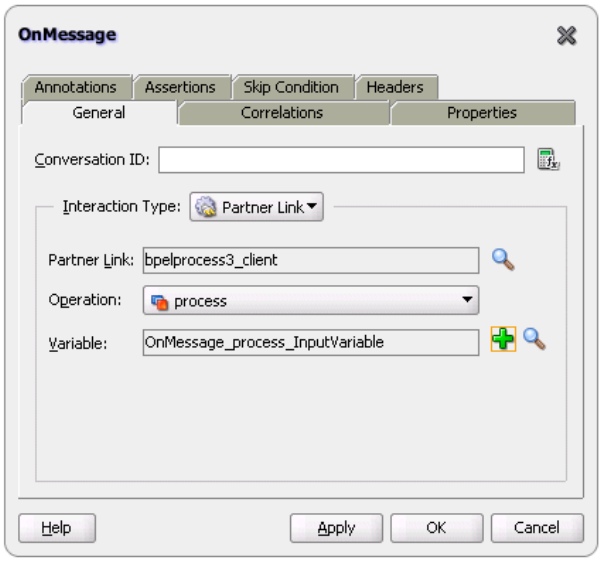
For more information, see Chapter 48, "Using Two-Layer Business Process Management (BPM)."

A.2.21 Pick Activity

This activity waits for the occurrence of one event in a set of events and performs the activity associated with that event. The occurrence of the events is often mutually exclusive (the process either receives an acceptance or rejection message, but not both). If multiple events occur, the selection of the activity to perform depends on which event occurred first. If the events occur nearly simultaneously, there is a race and the choice of activity to be performed is dependent on both timing and implementation.

The pick activity provides an OnMessage branch. When you double-click the **OnMessage** icon in BPEL 1.1, the dialog shown in Figure A–24 appears.

Figure A–24 OnMessage Dialog



In BPEL 2.0, the OnMessage dialog includes a **Documentation** tab and does not include a **Skip Condition** tab or an **Assertions** tab.

The two branches of the pick activity are as follows:

- onMessage (automatically displays below the **Pick** activity icon)

  Contains the code for receiving a reply, for example, from a loan service.

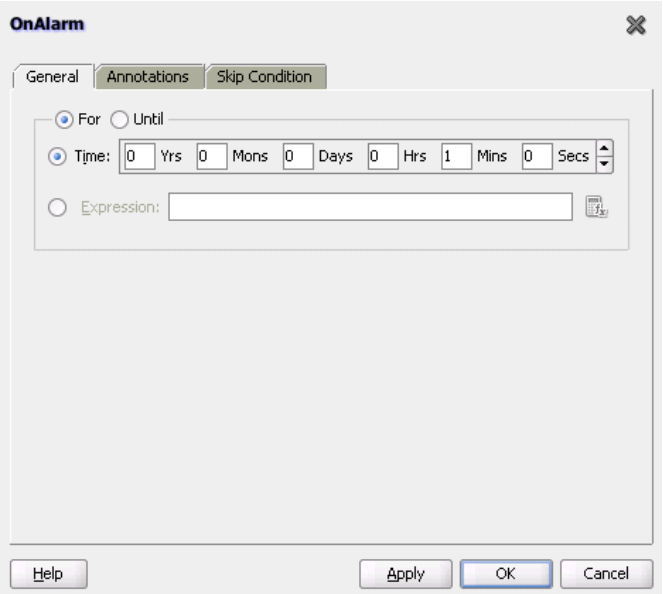
- onAlarm (does not automatically display; you must manually add this branch by selecting the **Pick** activity icon and clicking the **Add OnAlarm** icon)

  Contains the code for a timeout, for example, after one minute.

Whichever branch completes first is executed; the other branch is not executed. The branch that has its condition satisfied first is executed.

Figure A–25 shows the OnAlarm dialog of the pick activity in BPEL 1.1.

Figure A–25 OnAlarm Branch Dialog of a Pick Activity



In BPEL 2.0, the OnAlarm dialog includes a **Documentation** tab and does not include a **Skip Condition** tab.

> **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.

If you add correlations to an OnMessage branch, the correlations syntax is placed *after* the assign activity syntax. The correlation syntax must go *before* the assign activity.

As a work around, perform the following steps:

1. Create a correlation set in Oracle JDeveloper.
2. Assign this to the OnMessage branch.
3. Complete the remaining design tasks.
4. Before making or deploying the BPEL process, move the correlation syntax before the assign activity in the BPEL source code.

For more information about the pick activity, see the following:

- Chapter 5, "Introduction to Interaction Patterns in a BPEL Process"
- Section 6.17, "Mapping WSDL Message Parts in BPEL 2.0"
- Section 11.14, "Throwing Faults with Assertion Conditions"
- Section 14.2, "Creating a Pick Activity to Select Between Continuing a Process or Waiting"
- Section 14.6, "Setting Timeouts for Synchronous Processes"

A.2.22 Receive Activity

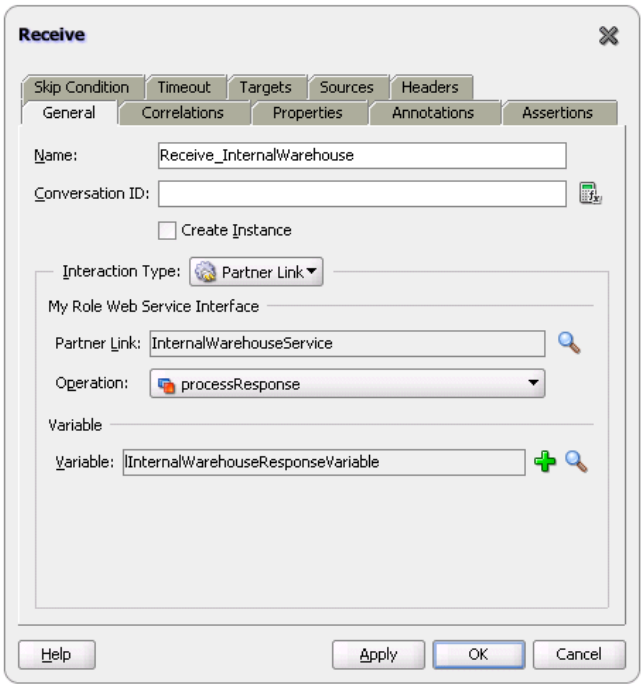
This activity specifies the partner link from which to receive information and the port type and operation for the partner link to invoke. This activity waits for an asynchronous callback response message from a service, such as a loan application approver service. While the BPEL process is waiting, it is dehydrated (compressed and stored) until the callback message arrives. The contents of this response are stored in a response variable in the process.

The receive activity supports the `bpelx:property` extensions that facilitate the passing of properties through the SOAP header, and the obtaining of SOA runtime system properties for useful information such as **tracking.compositeInstanceId** and **tracking.conversationId**.

Figure A–26 shows the Receive dialog in BPEL 1.1. You can perform the following tasks:

- Provide a meaningful name.
- Select the partner link service for which to specify an operation.
- Select the operation to be performed.
- Automatically create a variable or select an existing variable in which to transport the callback response.

Figure A–26 Receive Dialog



In BPEL 2.0, the Receive dialog includes a **Documentation** tab and does not include a **Skip Condition** tab, **Timeout** tab, or **Assertions** tab.

For more information about the receive activity, see the following:

- Chapter 5, "Introduction to Interaction Patterns in a BPEL Process"
- Section 6.17, "Mapping WSDL Message Parts in BPEL 2.0"
- Section 8.2.1.3, "Adding a Receive Activity"

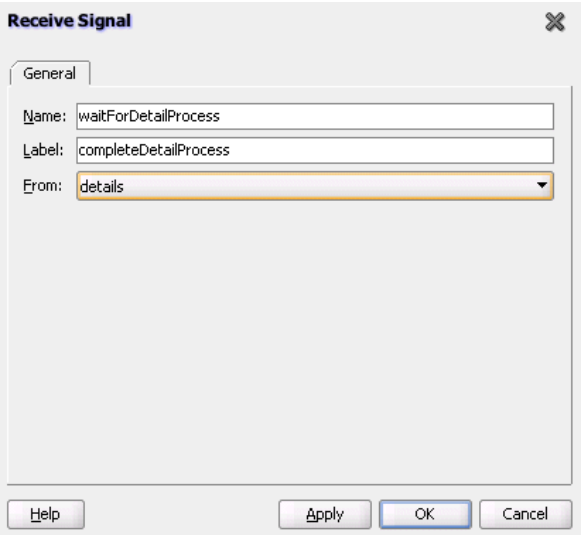
- Section 11.14, "Throwing Faults with Assertion Conditions"
- Section 14.3, "Setting Timeouts for Request-Response Operations in Receive Activities"

A.2.23 Receive Signal Activity

Use this activity in detail processes to wait for the notification signal from the master process to begin processing and use in a master process to wait for the notification signal from all detail processes indicating that processing has completed.

Figure A–27 shows the Receive Signal dialog in BPEL 1.1 and BPEL 2.0.

Figure A–27 Receive Signal Dialog



For more information, see Chapter 15, "Coordinating Master and Detail Processes."

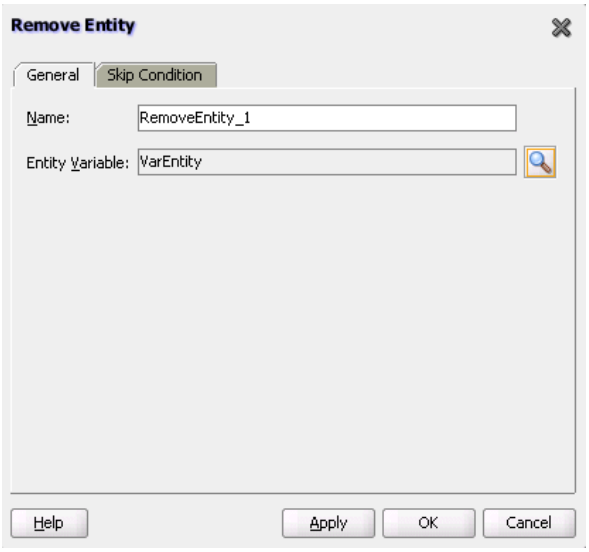
A.2.24 Remove Entity Activity

This activity enables you to remove an entity variable. This action removes the row.

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

Figure A–28 shows the Remove Entity dialog.

Figure A–28 Remove Entity Dialog

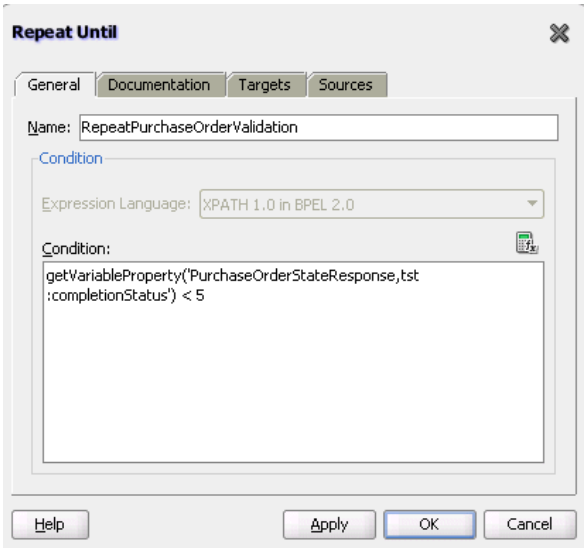


A.2.25 RepeatUntil Activity

Use this activity if the body of an activity must be performed at least once. 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. Figure A–29 shows the Remove Entity dialog.

Note: This activity is only supported in BPEL 2.0 projects.

Figure A–29 Repeat Until Dialog



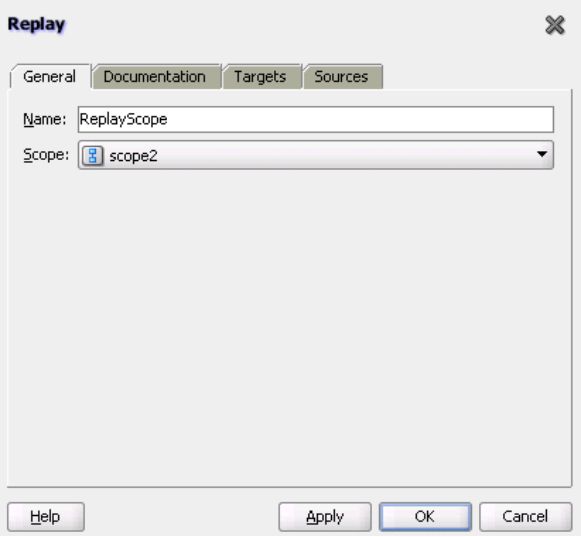
For more information about the repeatUntil activity, see, Section 10.4, "Creating a repeatUntil Activity to Define Conditional Branching."

A.2.26 Replay Activity

This activity enables you to re-execute the activities inside a selected scope.

Figure A–30 shows the Replay dialog in BPEL 2.0.

Figure A–30 Replay Dialog



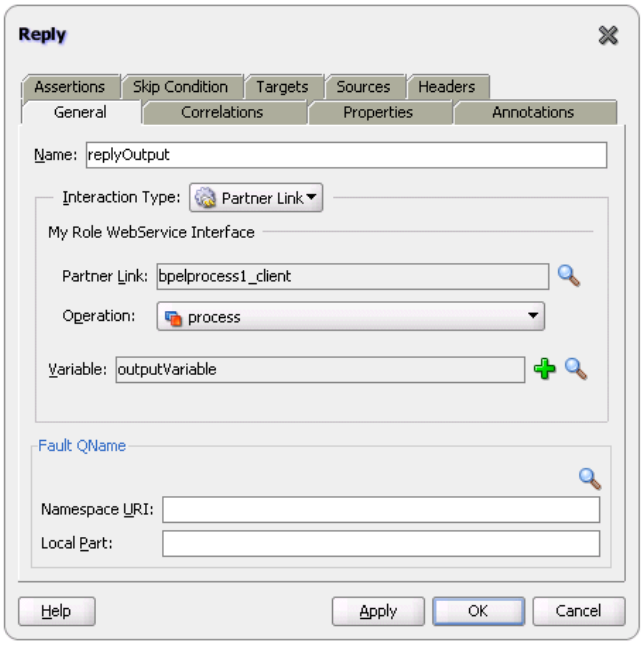
In BPEL 1.1, the Replay dialog includes a **Skip Condition** tab and does not include a **Documentation** tab, **Targets** tab, or **Sources** tab. For more information about the replay activity, see Section 11.11, "Re-executing Activities in a Scope Activity with the Replay Activity."

A.2.27 Reply Activity

This activity allows the process to send a message in reply to a message that was received through a receive activity. The combination of a receive activity and a reply activity forms a request-response operation on the WSDL port type for the process.

Figure A–31 shows the Reply dialog in BPEL 1.1.

Figure A–31 Reply Dialog



In BPEL 2.0, the Reply dialog includes a **Documentation** tab and does not include a **Skip Condition** tab or **Assertions** tab.

For more information about the reply activity, see the following:

- Chapter 5, "Introduction to Interaction Patterns in a BPEL Process"
- Section 6.17, "Mapping WSDL Message Parts in BPEL 2.0"
- Section 11.9.1, "How to Return a Fault in a Synchronous Interaction"

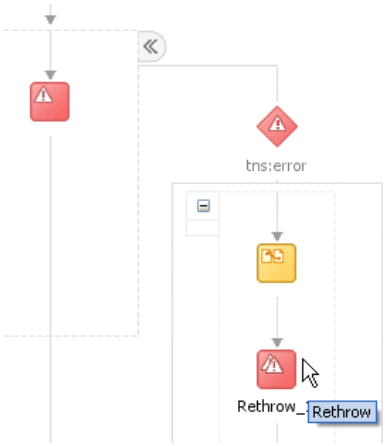
A.2.28 Rethrow Activity

This activity enables you to rethrow a fault originally captured by the immediately enclosing fault handler.

Note: This activity is only supported in BPEL 2.0 projects.

Figure A–32 shows a rethrow activity within a fault handler (catch activity).

Figure A–32 Rethrow Activity



For more information about rethrowing faults, see Section 11.8, "Rethrowing Faults with the Rethrow Activity."

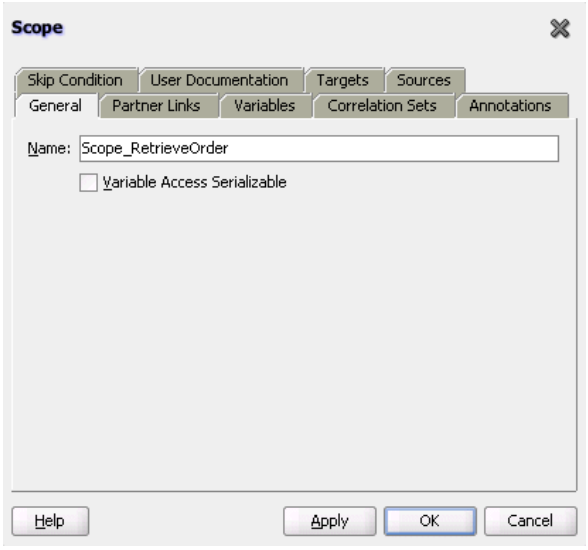
A.2.29 Scope Activity

This activity consists of a collection of nested activities that can have their own local variables, fault handlers, compensation handlers, and so on. A scope activity is analogous to a `{ }` block in a programming language.

Each scope has a primary activity that defines its behavior. The primary activity can be a complex structured activity, with many nested activities within it to arbitrary depth. The scope is shared by all the nested activities.

Figure A–33 shows the Scope dialog in BPEL 1.1. Define appropriate activities inside the scope activity.

Figure A–33 Scope Dialog



In BPEL 2.0, the Scope dialog includes a **Documentation** tab and does not include a **Skip Condition** tab.

Fault handling is associated with a scope activity. The goal is to undo the incomplete and unsuccessful work of a scope activity in which a fault has occurred. You define catch activities in a scope activity to create a set of custom fault-handling activities. Each catch activity is defined to intercept a specific type of fault.

Figure A–34 shows the **Add Catch** icon inside a scope activity. Figure A–35 shows the catch activity area that appears when you click the **Add Catch** icon. Within the area defined as **Drop Activity Here**, you drag additional activities to create fault handling logic to catch and manage exceptions.

For example, a client provides a social security number to a Credit Rating service when applying for a loan. This number is used to perform a credit check. If a bad credit history is identified or the social security number is identified as invalid, an assign activity inside the catch activity notifies the client of the loan offer rejection. The entire loan application process is terminated with a terminate activity.

Figure A–34 Creating a Catch Branch

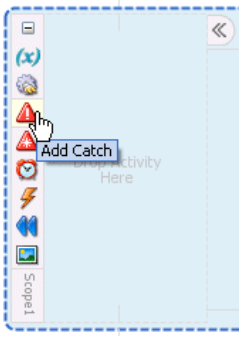
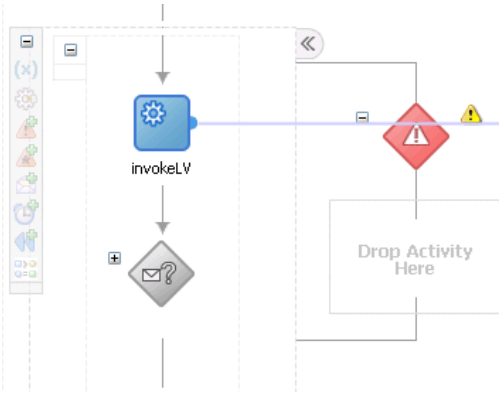


Figure A–35 Catch Activity Icon



For more information about the scope activity and fault handling, see the following:

- Chapter 5, "Introduction to Interaction Patterns in a BPEL Process"
- Section 6.17, "Mapping WSDL Message Parts in BPEL 2.0"
- Section 11.10, "Using a Scope Activity to Manage a Group of Activities"

A.2.30 Sequence Activity

This activity enables you to define a collection of activities to be performed in sequential order. For example, you may want the following activities performed in a specific order:

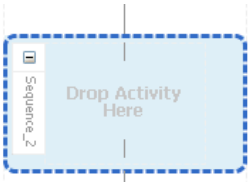
- A customer request is received in a receive activity.
- The request is processed inside a flow activity that enables concurrent behavior.
- A reply message with the final approval status of the request is sent back to the customer in a reply activity.

A sequence activity makes the assumption that the request can be processed in a reasonable amount of time, justifying the requirement that the invoker wait for a synchronous response (because this service is offered as a request-response operation).

When this assumption cannot be made, it is better to define the customer interaction as a pair of asynchronous message exchanges.

When you double-click the **Sequence** icon, the activity area shown in Figure A–36 appears. Drag and define appropriate activities inside the sequence activity.

Figure A–36 Sequence Activity



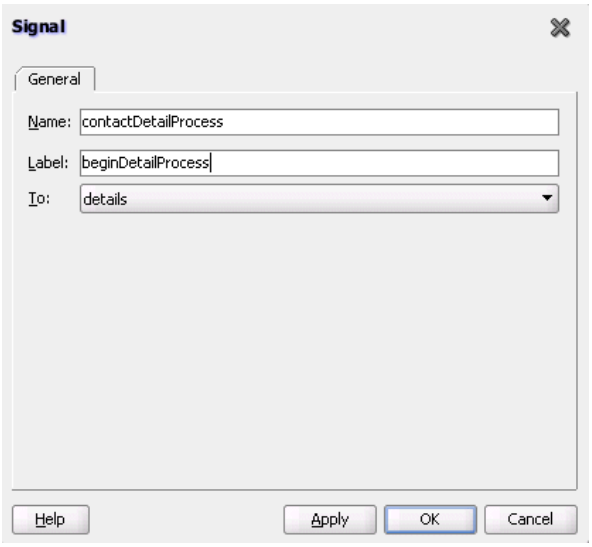
For more information about the sequence activity, see the following:

- Chapter 5, "Introduction to Interaction Patterns in a BPEL Process"
- Section 9.2, "Creating a Parallel Flow"

A.2.31 Signal Activity

This activity is used in a master process to notify detail processes to perform processing at runtime and used in detail processes to notify a master process that processing has completed. Figure A–37 shows the Signal dialog in BPEL 1.1 and BPEL 2.0.

Figure A–37 Signal Dialog



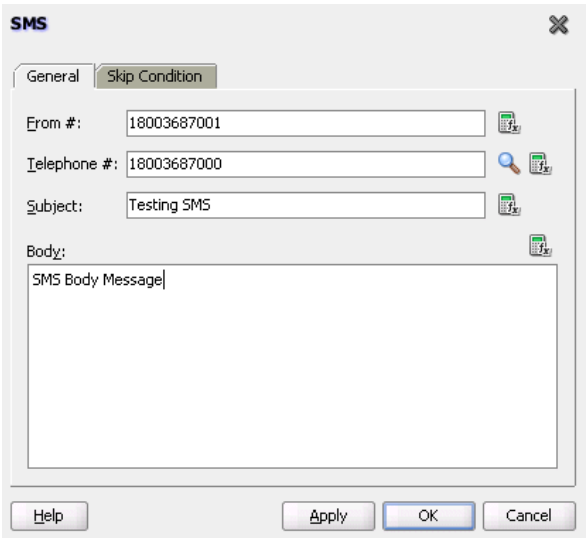
For more information, see Chapter 15, "Coordinating Master and Detail Processes."

A.2.32 SMS Activity

This activity enables you to send a short message system (SMS) notification about an event.

Figure A–38 shows the SMS dialog in BPEL 1.1.

Figure A–38 SMS Dialog



In BPEL 2.0, the SMS dialog does not include a **Skip Condition** tab.

For more information about the SMS activity, see Section 16.3.3, "How to Configure the SMS Notification Channel."

> **Note:** The fax and pager activities are not supported in 11*g*.

A.2.33 Switch Activity

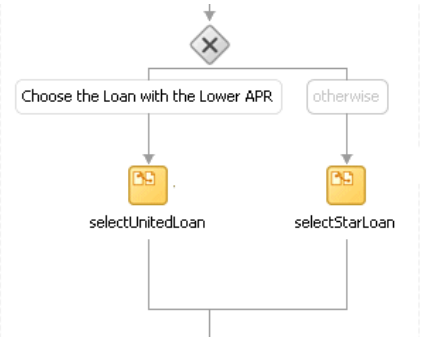
This activity consists of an ordered list of one or more conditional branches defined in a case branch, followed optionally by an otherwise branch. The branches are considered in the order in which they appear. The first branch whose condition is true is taken and provides the activity performed for the switch. If no branch with a condition is taken, then the otherwise branch is taken. If the otherwise branch is not explicitly specified, then an otherwise branch with an empty activity is assumed to be available. The switch activity is complete when the activity of the selected branch completes.

A switch activity differs in functionality from a flow activity. For example, a flow activity enables a process to gather two loan offers at the same time, but does not compare their values. To compare and make decisions on the values of the two offers, a switch activity is used. The first branch is executed if a defined condition (inside the case branch) is met. If it is not met, the otherwise branch is executed.

> **Note:** This activity is replaced by the if activity in BPEL 2.0 projects.

Figure A–39 shows a switch activity with the following defined branches.

Figure A–39 Switch Activity



For more information about the switch activity, see the following:

- Chapter 5, "Introduction to Interaction Patterns in a BPEL Process"
- Section 10.2.1, "Defining Conditional Branching with the Switch Activity in BPEL 1.1"

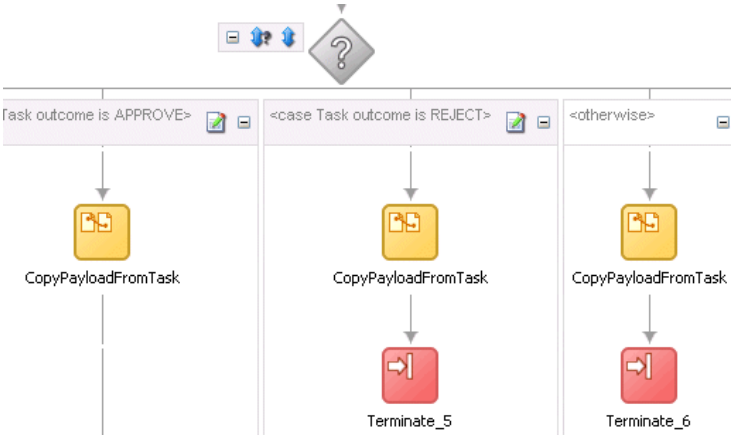
A.2.34 Terminate Activity

A terminate activity enables you to end the tasks of an activity (for example, the fault handling tasks in a catch branch). For example, if a client's bad credit history is identified or a social security number is identified as invalid, a loan application process is terminated, and the client's loan application document is never submitted to the service loan providers.

Note: This activity is replaced by the exit activity in BPEL 2.0 projects.

Figure A–40 shows several terminate activities in the otherwise branch of a switch activity.

Figure A–40 Terminate Activity



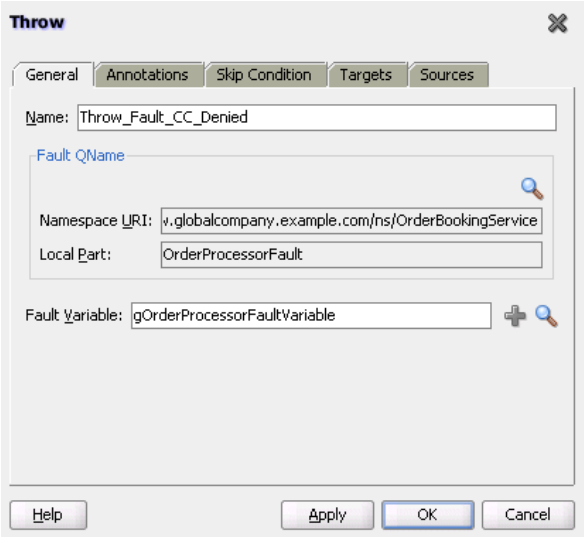
For more information about the terminate activity, see Section 11.13.1, "Stopping a Business Process Instance with the Terminate Activity in BPEL 1.1."

A.2.35 Throw Activity

This activity generates a fault from inside the business process.

Figure A–41 shows the Throw dialog.

Figure A–41 Throw Dialog



In BPEL 2.0, the Throw dialog includes a **Documentation** tab and does not include a **Skip Condition** tab.

For more information about the throw activity, see Section 11.7, "Throwing Internal Faults."

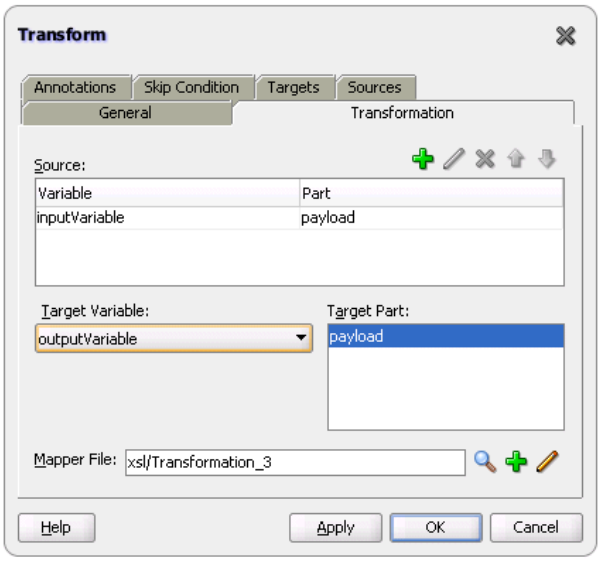
A.2.36 Transform Activity

This activity enables you to create a transformation that maps source elements to target elements (for example, incoming purchase order data into outgoing purchase order acknowledgment data).

Figure A–42 shows the Transform dialog in BPEL 1.1. This dialog enables you to perform the following tasks:

- Define the source and target variables and parts to map.
- Specify the transformation mapper file.
- Click the second icon (the **Add** icon) to the right of the **Mapper File** field to access the XSLT Mapper for creating a new XSL file for graphically mapping source and target elements. Click the **Edit** icon (third icon) to edit an existing XSL file.

Figure A–42 Transform Dialog



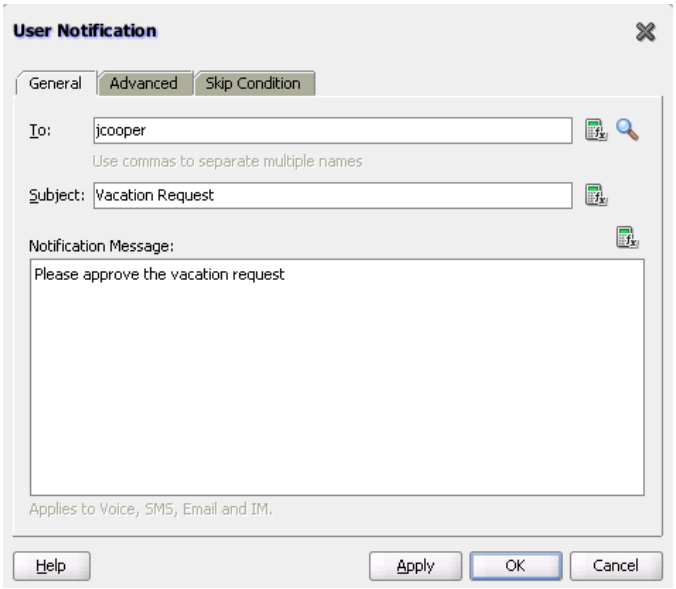
In BPEL 2.0, the Transform dialog includes a **Documentation** tab and does not include a **Skip Condition** tab.

For more information about the transform activity, see Chapter 37, "Creating Transformations with the XSLT Mapper."

A.2.37 User Notification Activity

This activity enables you to design a BPEL process in which you do not explicitly select a notification channel during design time, but simply indicate that a notification must be sent. The channel to use for sending notifications is resolved at runtime based on preferences defined by the end user in the User Messaging Preferences user interface of the Oracle User Messaging Service. This moves the responsibility of notification channel selection from Oracle BPEL Designer to the end user. If the end user does not select a preferred channel or rule, email is used by default for sending notifications to that user. Figure A–43 shows the User Notification dialog in BPEL 1.1.

Figure A–43 User Notification Dialog



In BPEL 2.0, the User Notification dialog does not include a **Skip Condition** tab.

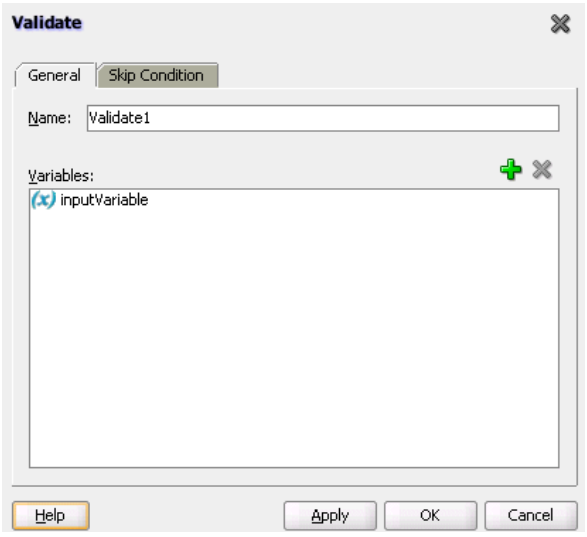
For more information, see Section 16.4, "Allowing the End User to Select Notification Channels."

A.2.38 Validate Activity

This activity enables you to validate variables in the list. The variables are validated against their XML schema.

Figure A–44 shows the Validate dialog in BPEL 1.1.

Figure A–44 Validate Dialog



In BPEL 2.0, the Validate dialog includes a **Documentation** tab, **Targets** tab, and **Sources** tab, and does not include a **Skip Condition** tab.

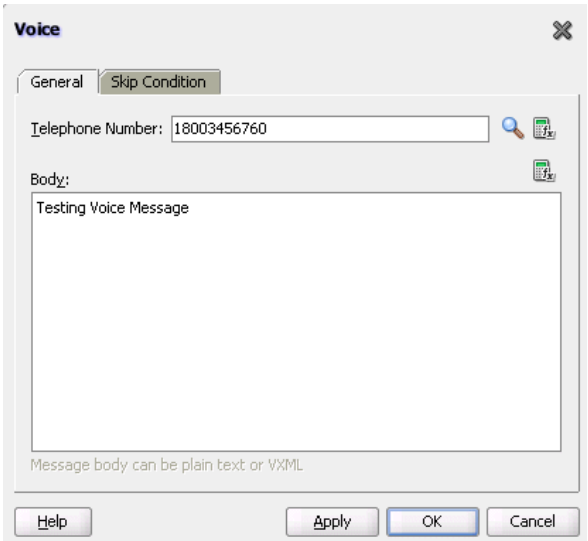
For more information about the validate activity, see Section 6.15, "Validating XML Data."

A.2.39 Voice Activity

This activity enables you to send a telephone voice notification about an event.

Figure A–45 shows the Voice dialog in BPEL 1.1.

Figure A–45 Voice Dialog



In BPEL 2.0, the Voice dialog does not include a **Skip Condition** tab.

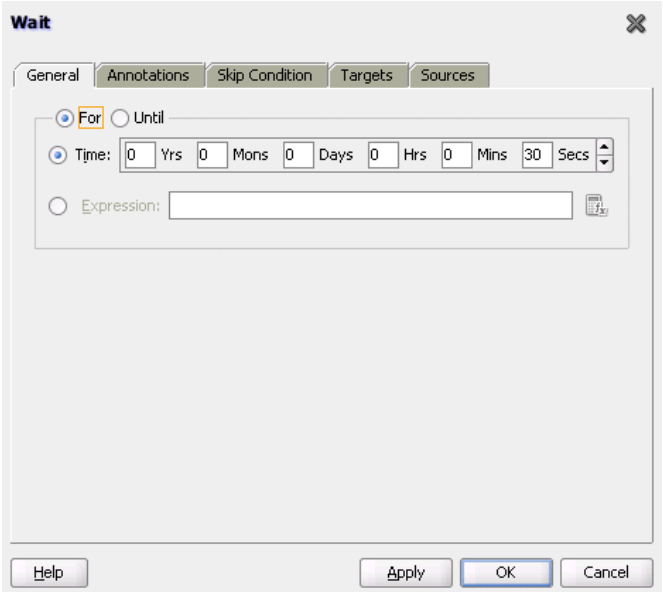
For more information about the voice activity, see Section 16.3.4, "How to Configure the Voice Notification Channel."

A.2.40 Wait Activity

This activity allows a process to specify a delay for a certain period or until a certain deadline is reached. A typical use of this activity is to invoke an operation at a certain time. This activity enables you to wait for a given time period or until a certain time has passed. Exactly one of the expiration criteria must be specified.

Figure A–46 shows the Wait dialog in BPEL 1.1.

Figure A–46 Wait Dialog



In BPEL 2.0, the Wait dialog includes a **Documentation** tab and does not include a **Skip Condition** tab.

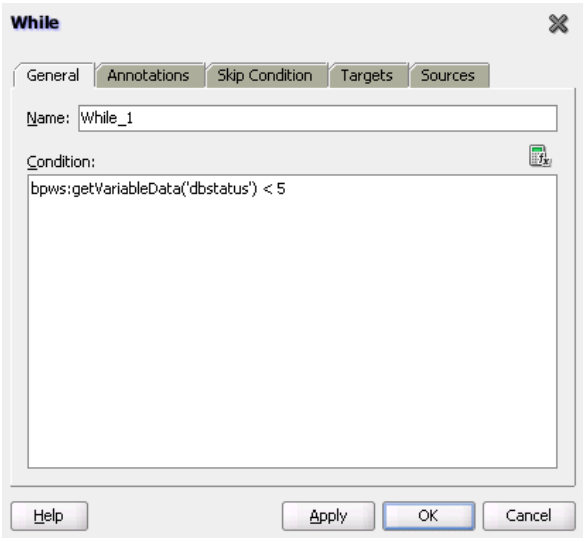
For more information about the wait activity, see Section 14.4, "Creating a Wait Activity to Set an Expiration Time."

A.2.41 While Activity

This activity supports repeated performance of a specified iterative activity. The iterative activity is repeated until the given `while` condition is no longer true.

Figure A–47 shows the While dialog in BPEL 1.1. You can enter expressions in this dialog.

Figure A–47 While Dialog



In BPEL 2.0, the While dialog includes a **Documentation** tab and does not include a **Skip Condition** tab.

For more information about the while activity, see Section 10.3, "Creating a While Activity to Define Conditional Branching."

A.3 Introduction to BPEL Services

BPEL processes can communicate with web-based applications and clients through web services, Oracle Application Development Framework (ADF)-business component (BC) services, JCA adapters, Oracle B2B services, Oracle Business Activity Monitoring, HTTP binding, direct binding, EJB services, Oracle E-Business Suite, and partner links.

To access BPEL services:

1. In the Component Palette of Oracle BPEL Designer, expand **BPEL Services** to display the services.

For more information about the adapters described in the following sections, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

A.3.1 ADF-BC Service

This service connects Oracle ADF applications using SDOs with the SOA platform.

A.3.2 AQ Adapter

This adapter acts as both a dequeue (inbound) and enqueue (outbound) messaging adapter. In the inbound direction, the adapter polls the queues for messages to dequeue from a destination. In the outbound direction, the adapter enqueues messages to the queue for subscribers to dequeue.

A.3.3 Oracle B2B

This adapter enables you to browse B2B metadata in the Metadata Service (MDS) repository and select document definitions.

Oracle B2B is an e-commerce gateway that enables the secure and reliable exchange of transactions between an organization and its external trading partners. Oracle B2B and Oracle SOA Suite are designed for e-commerce business processes that require process orchestration, error mitigation, and data translation and transformation within an infrastructure that addresses the issues of security, compliance, visibility, and management.

For more information, see *Oracle Fusion Middleware User's Guide for Oracle B2B*.

A.3.4 Oracle BAM Adapter

This adapter integrates Java EE applications with Oracle BAM Server to send data. This adapter is used as a reference binding component in a SOA composite application.

For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* and Part X, "Using Oracle Business Activity Monitoring".

A.3.5 Database Adapter

This adapter enables a BPEL process to communicate with Oracle databases or third-party databases through JDBC. To access an existing relational schema, you use the Adapter Configuration Wizard to do the following:

- Import a relational schema and map it as an XML schema (XSD).
- Abstract SQL operations such as `SELECT`, `INSERT`, and `UPDATE` as web services.

While your BPEL process deals with XML and invokes web services, database rows and values are queried, inserted, and updated.

A.3.6 Direct Binding Service

This service uses the Direct Binding API to invoke a SOA composite application in the inbound direction and exchange messages over a remote method invocation (RMI). This option supports the propagation of both identities and transactions across JVMs and uses the T3 optimized path. Both synchronous and asynchronous invocation patterns are supported.

You can also invoke an Oracle Service Bus (OSB) flow or another SOA composite application in the outbound direction.

For more information about the Direct Binding Invocation API, see *Oracle Fusion Middleware Infrastructure Management Java API Reference for Oracle SOA Suite* and Chapter 36, "Using the Direct Binding Invocation API."

For more information about OSB, see *Oracle Fusion Middleware Developer's Guide for Oracle Service Bus*.

A.3.7 EJB Service

This service enables you to send and receive messages through Enterprise JavaBeans (EJBs).

For more information, see Chapter 35, "Integrating Enterprise JavaBeans with SOA Composite Applications."

A.3.8 File Adapter

This adapter acts as both an inbound and outbound adapter. In the inbound direction, the adapter polls for files in a directory to retrieve and process. In the outbound direction, the adapter creates files in a directory.

A.3.9 FTP Adapter

This adapter acts as both an inbound and outbound adapter. In the inbound direction, the adapter polls for files in a directory to retrieve and process. In the outbound direction, the adapter creates files in a directory.

A.3.10 HTTP Binding

This service enables you to integrate SOA composite applications with HTTP binding. This service enables you to invoke SOA composite applications through HTTP POST and GET operations, and invoke HTTP endpoints through HTTP POST and GET operations.

For more information, see Section 34.1.2, "HTTP Binding Service."

A.3.11 JMS Adapter

This adapter acts as both a consume (inbound) and produce (outbound) messaging adapter. In the inbound direction, the adapter polls (consumes) messages from a JMS destination. In the outbound direction, the adapter sends (produces) messages to a JMS destination.

A.3.12 MQ Adapter

This adapter provides message exchange capabilities between BPEL processes and the IBM MQSeries messaging software.

A.3.13 Oracle Applications

This adapter provides comprehensive, bidirectional, multimodal, synchronous, and asynchronous connectivity to Oracle Applications. The adapter supports all modules of Oracle Applications in Release 12 and Release 11*i*, including selecting custom integration interface types based on the version of Oracle E-Business Suite. The adapter provides real-time and bidirectional connectivity to Oracle Applications through interface tables, views, application programming interfaces (APIs), and XML Gateway. The adapter inserts data into Oracle Applications using interface tables and APIs. To retrieve data from Oracle Applications, the adapter uses views. In addition, it uses XML Gateways for bidirectional integration with Oracle Applications. XML Gateways are also used to insert and receive Open Application Group Integration Specification (OAGIS)-compliant documents from Oracle Applications.

A.3.14 Socket Adapter

This adapter enables you to model standard or nonstandard protocols for communication over TCP/IP sockets. You can use this adapter to create a client or server socket, and establish a connection. The data that is transported can be text or binary.

A.3.15 Third Party Adapter

This adapter enables you to integrate third-party adapters into a SOA composite application. These third-party adapters produce artifacts (WSDLs and JCA files) that can configure a JCA adapter.

A.3.16 Web Service

This service enables you to connect to standards-based services using SOAP over HTTP.

For more information, see Section 2.3, "Adding Service Binding Components."

A.4 Publishing and Browsing the Oracle Service Registry

The Oracle Service Registry (OSR) provides a common standard for publishing and discovering information about web services. This section describes how to configure OSR against a separately installed Oracle SOA Suite environment.

You can use Oracle SOA Suite with the following versions of OSR:

- OSR 11*g*
- OSR 10.3 (with Oracle WebLogic Server 10.3)

- OSR 10.1.3

For more information about OSR, visit the following URL:

http://www.oracle.com/technetwork/middleware/registry/overview/index.html

Notes:

- This section does *not* describe how to configure OSR against the embedded Oracle WebLogic Server in Oracle JDeveloper.
- OSR 10.3 deploys to the 10.3.0.0 version of Oracle WebLogic Server.
- OSR 10.3 does not support the 10.3.1.0 version of Oracle WebLogic Server.

A.4.1 How to Publish a Business Service

This section provides an overview of how to publish a business service. For specific instructions, see the documentation at the following URL:

http://www.oracle.com/technetwork/middleware/registry/overview/index.html

You can also access the documentation by clicking the **Registry Documentation** icon in the upper right corner of the page.

To publish a business service:

1. Go to the Registry Control:

   `http://hostname:port/registry/uddi/web`

2. Click **Publish** > **WSDL**.
3. Log in when prompted.
4. Complete the fields on this page to specify the access point URL and publish the WSDL for the business service.

Note: If you later change your endpoint location, you must also update the WSDL location in the Registry Control. Otherwise, UDDI invocation fails during runtime. See section Section A.4.4.1, "Changing Endpoint Locations in the Registry Control."

A.4.2 How to Create a Connection to the Registry

To create a connection to the registry:

1. Go to Oracle JDeveloper.
2. Select **File** > **New** > **Connections** > **UDDI Registry Connection** to create a UDDI connection.
3. Enter a connection name.
4. Enter an inquiry endpoint URL. For example:

   `http://myhost.us.oracle.com:7001/registry/uddi/inquiry`

5. Ensure that the **Business View** option is selected.
6. Click **Next**.
7. Click **Test Connection**.
8. If successful, click **Finish**. Otherwise, click the **Back** button and correct your errors.

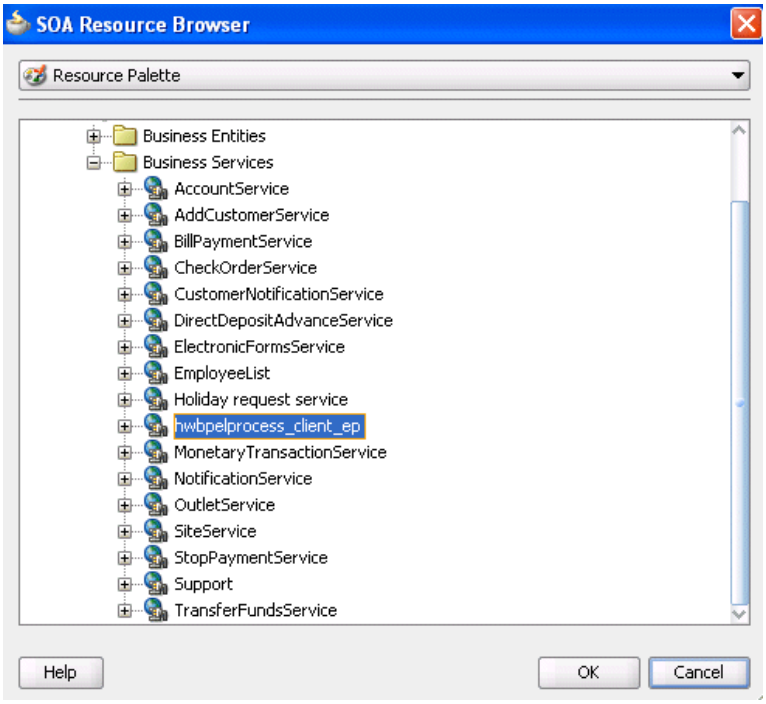
A.4.3 How to Configure a SOA Project to Invoke a Service from the Registry

To configure a SOA project to invoke a service from the registry:

1. Open the SOA project in which to create a reference to the business service.
2. Drag a **Web Service** icon into the **External References** swimlane.

   The Create Web Service dialog appears.
3. To the right of the **WSDL URL** field, click the icon to select a WSDL.
4. From the list at the top, select **Resource Palette**.
5. Expand the navigational tree.
6. Expand **UDDI Registry** > **Business Services**.
7. Select the published business service, and click **OK**. Figure A–48 provides details.

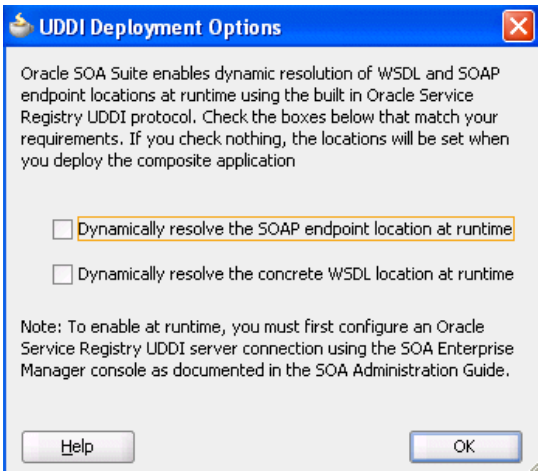
Figure A–48 Business Service



   The UDDI Deployment Options dialog appears.
8. Select one of the following deployment options:
   - **Dynamically resolve the SOAP endpoint location at runtime**
   - **Dynamically resolve the concrete WSDL location at runtime**

Figure A–49 provides details.

Figure A–49 UDDI Deployment Options Dialog



9. Click **OK**.

   You are returned to the Create Web Service dialog.

10. See the following section based on your selection in the UDDI Deployment Options dialog.

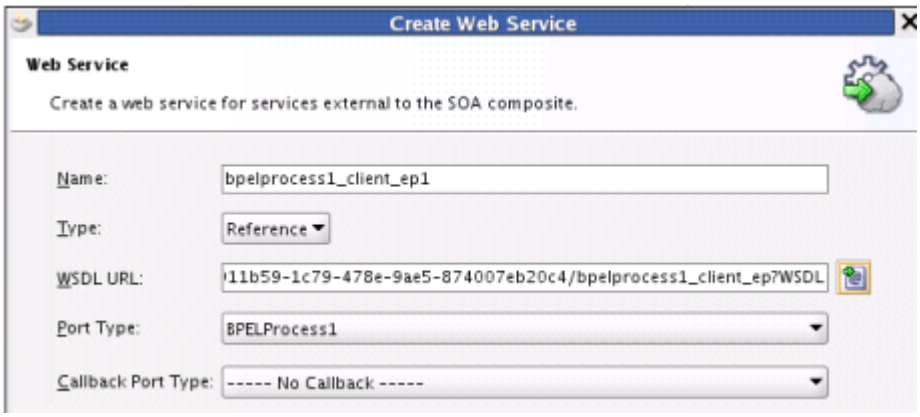
    - Section A.4.3.1, "Dynamically Resolving the SOAP Endpoint Location"
    - Section A.4.3.2, "Dynamically Resolving the WSDL Endpoint Location"

A.4.3.1 Dynamically Resolving the SOAP Endpoint Location

1. Complete the remaining fields in the Create Web Service dialog, and click **OK**.

   The Create Web Service dialog looks as shown in Figure A–50.

Figure A–50 Create Web Service Dialog - SOAP Endpoint Location



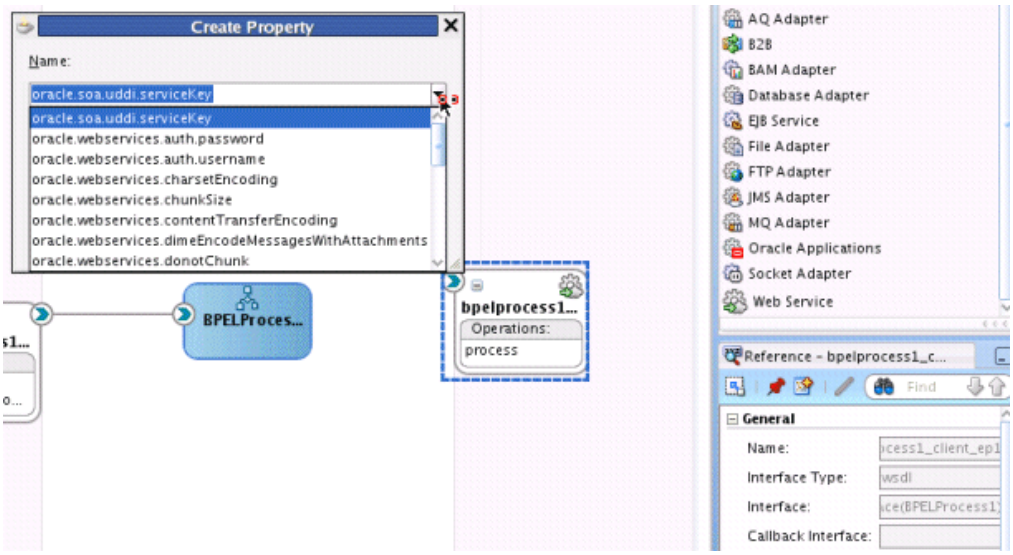
2. Wire the reference with the appropriate service component.
3. In the SOA Composite Editor, click **Source**.

   The `composite.xml` file shows the `serviceKey`. The property dynamically resolves the endpoint binding location at runtime.

```
<property name="oracle.soa.uddi.servicekey" type="xs:string" many="false">uddi:
 d3611b59-1c79-478e-9ae5-874007eb20c4">
```

4. If you want, you can also resolve the SOAP endpoint location by explicitly adding the `oracle.soa.uddi.servicekey` property in the Property Inspector. This action dynamically resolves the SOAP endpoint location at runtime for any external reference to a web service. Figure A–51 provides details.
   a. Highlight the reference binding component in the **External References** swimlane.
   b. In the **Property Inspector**, expand the **Properties** section.
   c. Click the **Add** icon.
   d. In the **Name** list, select **oracle.soa.uddi.servicekey**.
   e. In the **Value** field, specify the value for **oracle.soa.uddi.servicekey** from the `composite.xml` file.

Figure A–51 serviceKey Properties

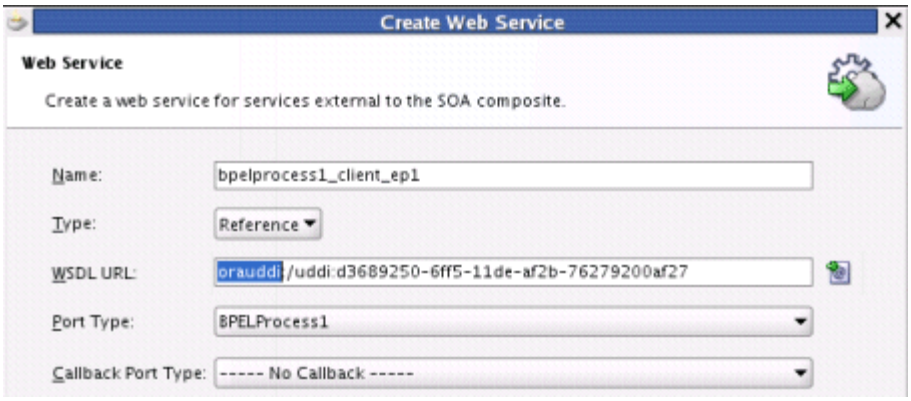


A.4.3.2 Dynamically Resolving the WSDL Endpoint Location

1. Complete the remaining fields in the Create Web Service dialog, and click **OK**.

   The Create Web Service dialog looks as shown in Figure A–52.

Figure A–52 Create Web Service Dialog - WSDL Endpoint Location



2. Wire the reference with the appropriate service component.
3. In the SOA Composite Editor, click **Source**.

   The `composite.xml` file shows that the WSDL location is an abstract URL of `orauddi:/`*`uddi_service_key`* instead of a concrete URL (such as a HTTP URL). The `orauddi` protocol dynamically resolves the WSDL location at runtime.

```
<location="orauddi:/uddi:d3689250-6ff5-11de-af2b-76279200af27">
```

A.4.3.3 Resolving Endpoints

Oracle SOA Suite invokes a service for resolving an endpoint. Examples and descriptions are shown in Table A–2.

Table A–2 Resolving Endpoints

Endpoint Resolutions	Description	Example
Normalized message UDDI `serviceKey`	The OSR UDDI `serviceKey` is specified in the normalized message property within an Oracle Mediator or an Oracle BPEL Process Manager assign activity (`serviceKey`).	For example, with Oracle Mediator: `<copy target="$out.property.oracle.soa.uddi.serviceKey" value="uddi:10a55fa0-99e8-11df-9edf-7d5e3ef09eda"/>`
Normalized message `endpointURI`	The normalized message `endpointURI` property is specified within an Oracle Mediator or an Oracle BPEL Process Manager assign activity (`endpointURI`).	For example, with Oracle Mediator: `<copy target="$out.property.endpointURI" value="http://`*`hostname`*`:8001/soa-infra/services /partition/Project/endpoint_ep"/>`

Table A–2 (Cont.) Resolving Endpoints

Endpoint Resolutions	Description	Example
`composite.xml` UDDI `serviceKey`	The OSR UDDI `serviceKey` property (`oracle.soa.uddi.serviceKey`) is specified in the binding component section of `composite.xml`. **Note:** This can be overwritten in Oracle Enterprise Manager Fusion Middleware Control.	`<binding.ws port="http://xmlns.oracle.com/UDDIPublishApplication /Proj/BPELProcess1#wsdl.endpoint(bpelprocess1_client _ep/BPELProcess1_pt)" . . .> <property name="oracle.soa.uddi.serviceKey" type="xs:string" many="false">uddi:31040650-9ce7-11df-9ee1-7d5e3e f09eda</property> </binding.ws>`
`composite.xml` `endpointURI`	The `endpointURI` property is specified within the binding component section of `composite.xml`. **Note:** This can be overwritten in Oracle Enterprise Manager Fusion Middleware Control.	`<binding.ws port="http://xmlns.oracle.com/UDDIPublishApplica tion/Project/BPELProcess1#wsdl.endpoint(bpelproc ess1_client_ep/BPELProcess1_pt)" . . . > <property name="oracle.soa.uddi.endpointURI" value="http://hostname:8001/soa-infra/services/ Partition/Project/bpelprocess1_client_ep"</property> </binding.ws>`
`composite.xml` concrete WSDL endpoint location	The endpoint location is specified in the concrete WSDL in the binding component section of `composite.xml`.	`<binding.ws port="http://xmlns.oracle.com/UDDIPublishApplication /Project/BPELProcess1#wsdl.endpoint(bpelprocess1_ client_ep/BPELProcess1_pt)" location="http://hostname:8001/soa-infra/services /Partition/Project/bpelprocess1_client_ep?wsdl" soapVersion="1.1">`

The failover scenario for resolving endpoints is as follows.

- Normalized message UDDI `serviceKey`
  - Any error on the endpoint access
    - Log a severe error
    - Return an error to the user
- Normalized message `endpointURI`
  - Any error on the endpoint access
    - Log a severe error
    - Return an error to the user
- `composite.xml` UDDI `serviceKey`
  - Error on an OSR connection
    - Log a severe error
    - Use the `composite.xml` `endpointURI` if it is coded
    - Else, return an error to the user
  - Error for an invalid `serviceKey` in the connection
    - Log a severe error
    - Use the `composite.xml` `endpointURI` if it is coded

        * Else, return an error to the user
      – Error on the endpoint access
        * Log a warning error
        * Use a second (or third) binding template if it exists.
        * Else, fail over to the `composite.xml endpointURI`
- `composite.xml endpointURI`
  – Error on the endpoint access
    * Log a warning error
    * Fail over to the `composite.xml` concrete WSDL endpoint location
- `composite.xml` concrete WSDL endpoint location
  – Error on the endpoint access
    * Log a severe error
    * Return an error to the user

A.4.4 How To Configure the Inquiry URL, UDDI Service Key, and Endpoint Address for Runtime

You can set the inquiry URL, UDDI service key, and endpoint address during runtime in Oracle Enterprise Manager Fusion Middleware Control.

To configure the inquiry URL, service key, and endpoint reference for runtime:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control.
2. Specify values for the following properties:
   - In the SOA Infrastructure Common Properties page, specify the same UDDI inquiry URL that you specified in the Create UDDI Registry Connection wizard. For information, see section "Configuring SOA Infrastructure Properties" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.
   - In the Properties page of the reference binding component, you can change the endpoint reference and service key values created during design time. For information, see section "Configuring Service and Reference Binding Component Properties" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.
3. Restart the SOA Infrastructure.
4. Exit Oracle Enterprise Manager Fusion Middleware Control.
5. To see endpoint statistics, return to the Registry Control.
6. Go to the Manage page and check statistics to see the increase in the number of invocations when not cached (the first time).

   Caching of WSDL URLs occurs by default during runtime. If a WSDL URL is resolved using the orauddi protocol, subsequent invocations retrieve the WSDL URLs from cache, and not from OSR. When an endpoint WSDL obtained from cache is no longer reachable, the cache is refreshed and OSR is contacted to retrieve the new endpoint WSDL location. As a best practice, Oracle recommends that you undeploy services that are no longer required in Oracle Enterprise Manager Fusion Middleware Control and used by the SOA Infrastructure.

Endpoint services that are shut down or retired (but not undeployed) are still reachable. Therefore, the cache is not refreshed.

If you move the business service WSDL from one host to another, ensure that you change the location in the Registry Control. No change is required in Oracle JDeveloper or Oracle Enterprise Manager Fusion Middleware Control.

You can optionally increase the amount of time that the WSDL URL is available in cache for inquiry by the service key. For more information, see "Configuring Service and Reference Binding Component Properties" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

Note: In 11*g*, caching occurs automatically. If you are using Oracle SOA Suite 10.1.3, caching is supported by setting the `CacheRegistryWSDL` property to `true` in `bpel.xml`. Setting this property to `false` disables caching.

A.4.4.1 Changing Endpoint Locations in the Registry Control

The Registry Control provides an option for changing the endpoint location. This is a two-step process. The following steps provide an overview. For more specific details, see the Oracle Service Registry documentation:

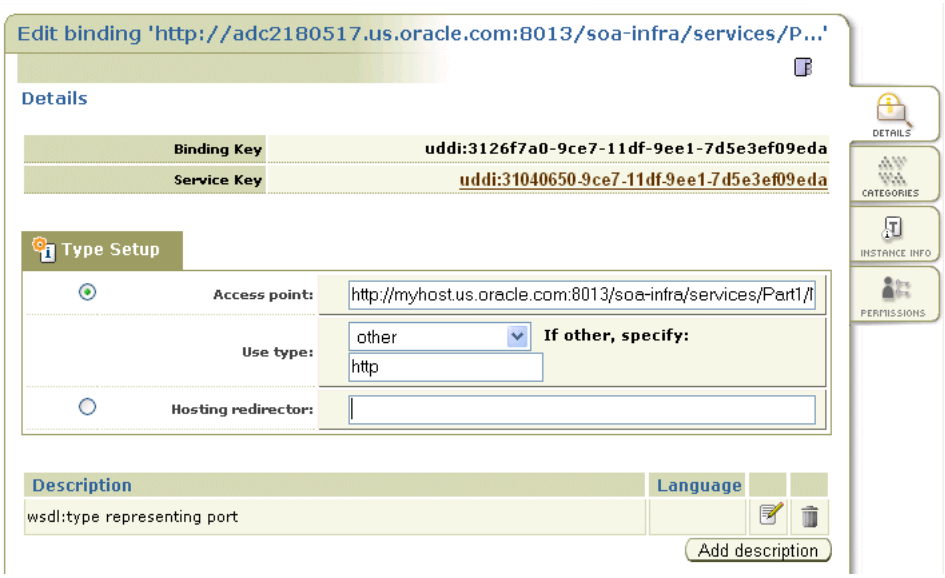
`http://www.oracle.com/technetwork/middleware/registry/overview/index.html`

To update WSDL bindings:

1. Log in to Registry Control.
2. Click **Search** > **Business**.
3. Click **Add Name**.
4. In the **Name** field, enter a search criteria.
5. Click **Find**.
6. In the search results, click the business name that is displayed.
7. On the right side, click the **Services** tab.
8. Click the service name from the list of services.
9. At the bottom, click the **Edit** button.
10. On the right side, click the **Bindings** tab.
11. In the list of bindings, select the **notepad** icon next to the description column.

    Oracle Service Registry is now in edit mode for bindings.
12. In the **Access Point** field, change the required URL, and save your changes. Figure A–53 provides details.

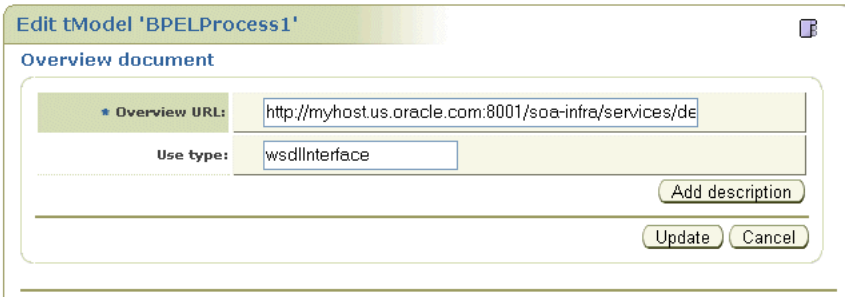
Figure A–53 Service and Binding Changes



To update WSDL binding overview documentation:

1. Within the Registry Control, click **Search**.
2. In the **tModel name** field, enter the name and click **Find tModel**.
3. In the **name** column, click the name with the description **wsdl:type representing portType**.
4. Ensure that WSDL details are shown correctly.
5. Click the **Edit** button.
6. On the right side, click the **Overview doc** tab.
7. Under the **Add description** button, click the **Edit** icon.
8. Enter the new URL.
9. Click **Update** and save the changes. Figure A–54 provides details.

Figure A–54 WSDL URL Verification



10. To verify, navigate to the service and ensure that the WSDL URL is pointing to a new location.

A.4.4.2 Publishing WSDLs from Multiple SOA Partitions

Follow these steps if you want to publish WSDLs from multiple SOA partitions using the Registry Control, and access them using a separate `serviceKey` and bindings.

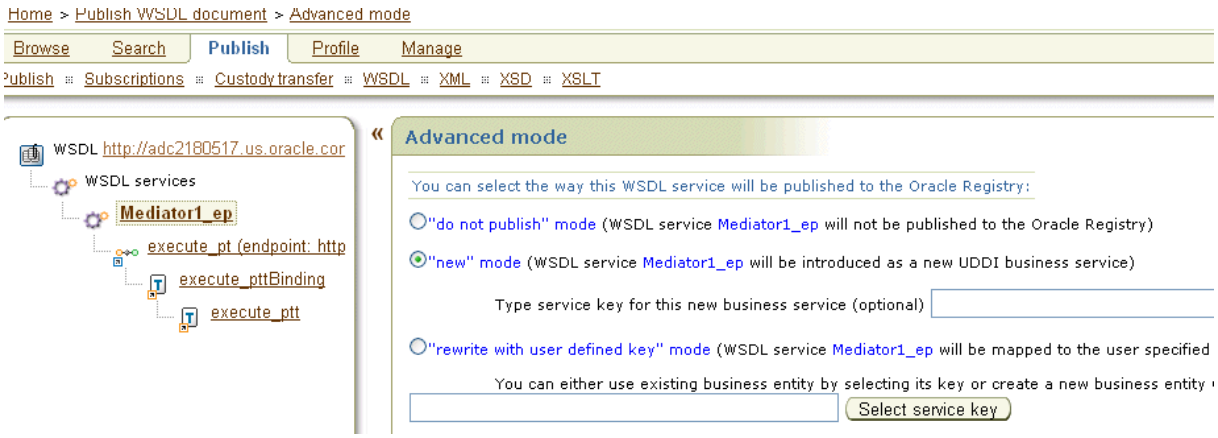
To publish WSDLs from multiple SOA partitions:

1. Log in to Registry Control.

   `http://host:port/registry/uddi/web`

2. Publish the WSDL from the first partition.
3. Publish the WSDL from the second partition.
   a. Click **Publish** > **WSDL**.
   b. Enter values in the **Business key** and **WSDL location (URI)** fields.
   c. Select the **Advanced Mode** checkbox.
   d. Click **Publish**.
   e. In the navigation tree in the left pane, select the endpoint, bindings, and port type, and ensure that the **"new" mode** option is selected. Figure A–55 provides details.

Figure A–55 Advanced Mode



   f. Click **Publish**.

A.4.5 How to Publish WSDLs to UDDI for Multiple Partitions

The following limitations exist for publishing WSDL services from Oracle Enterprise Manager Fusion Middleware Control.

- You cannot publish the same service with the same target namespace from different SOA partitions or from different hosts.
- There is no option for entering your own service key.

Instead, use the Registry Console to publish the same WSDL service deployed to different partitions to OSR.

To publish WSDLs to UDDI for multiple partitions:

1. Log in to the Registry Console.
2. Publish the WSDL of the first partition.

3. Rename the above-mentioned service name to a unique name.
4. Publish the WSDL of the second partition.

   This creates two separate services in OSR.

A.5 Providing Design-time Governance with the Oracle Enterprise Repository

The Oracle Enterprise Repository provides design-time governance in support of the service life cycle, delivering capabilities for the storage and management of metadata for composites, services, business processes, and other IT-related assets.

Oracle Enterprise Repository acts as the central source of Oracle SOA Suite information, providing all participants in the service life cycle with a human-centric discovery environment for planned, existing, and retired services.

Oracle Enterprise Repository provides role-based links to the artifact stores of the assets that it describes and links to design documents, justification documents, test plans, support plans, policies, and other forms of documentation.

From an integrated development environment (IDE) such as Oracle JDeveloper, you can perform the following tasks:

- Harvest Oracle SOA Suite project artifacts, including BPEL, WSDL, XSD, and XSLT files and file directories. After harvesting, the Oracle Enterprise Repository automatically creates assets, populates asset metadata, and generates relationship links based on the information in the artifact files.
- Browse for assets and artifacts available in the Oracle Enterprise Repository.
- View asset details such as description, usage history, expected savings, and relationships.
- Download an asset's artifacts (that is, payload) into your project. Typically, an asset payload is the functionality that you need for using a service (such as a WSDL file) or incorporating it into your code base (such as a binary or a BPEL file).
- Consume a WSDL file or a service from the Oracle Enterprise Repository.

For more information about these tasks and how to configure and use Oracle Enterprise Repository with an IDE, see the *Oracle Fusion Middleware Integration Guide for Oracle Enterprise Repository*.

For more information about harvesting from Oracle JDeveloper, see the *Oracle Fusion Middleware Configuration Guide for Oracle Enterprise Repository*.

A.6 Validating When Loading a Process Diagram

You may see an icon (a yellow triangle with an exclamation point) indicating invalid settings as you create and open activities such as a scope or an assign for the first time. The settings are invalid because you have not yet entered details.

To turn this option off for the current project, do the following:

1. Right-click the BPEL diagram and select **Display > Diagram Properties**.
2. Deselect the **Enable Automatic Validation** option.
3. Click **OK**.
4. Select **Save All** from the **File** main menu.

B

XPath Extension Functions

This appendix describes the XPath extension functions. Oracle provides XPath functions that use the capabilities built into Oracle SOA Suite and XPath standards for adding new functions.

This appendix includes the following sections:

- Section B.1, "SOA XPath Extension Functions"
- Section B.2, "BPEL XPath Extension Functions"
- Section B.3, "Oracle Mediator XPath Extension Functions"
- Section B.4, "Advanced Functions"
- Section B.5, "Workflow Service Functions"
- Section B.6, "Building XPath Expressions in Oracle JDeveloper"
- Section B.7, "Creating User-Defined XPath Extension Functions"

For additional information about XPath functions, visit the following URL:

`http://www.w3.org`

B.1 SOA XPath Extension Functions

This section describes the following types of SOA XPath extension functions:

- Database functions
- Date functions
- Mathematical functions
- String functions

B.1.1 Database Functions

This section describes the following database functions:

B.1.1.1 lookup-table

This function returns a string based on the SQL query generated from the parameters.

The string is obtained by executing:

```
SELECT outputColumn FROM table WHERE inputColumn = key
```

against the data source that can be either a JDBC connect string (`jdbc:oracle:thin:`*`username`*`/`*`password`*`@`*`host`*`:`*`port`*`:`*`sid`*) or a data source

JNDI identifier. Only the Oracle thin driver is supported if the JDBC connect string is used.

Example: `oraext:lookup-table('employee','id','1234','last_name','jdbc:oracle:thin:xyz/xyz@localhost:1521:ORCL')`

Signature:

```
oraext:lookup-table(table, inputColumn, key, outputColumn, data source)
```

Arguments:

- `table` - The table from which to draw the data.
- `inputColumn` - The column within the table.
- `key` - The key value of the input column.
- `outputColumn` - The column to output the data.
- `data source` - The source of the data.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.1.1.2 query-database

This function returns a node set by executing the SQL query against the specified database.

Signature:

```
oraext:query-database(sqlquery as string, rowset as boolean, row as boolean, data source as string)
```

Arguments:

- `sqlquery` - The SQL query to perform.
- `rowset` - Indicates if the rows should be enclosed in an element.
- `row` - Indicates if each row should be enclosed in an element.
- `data source` - Either a JDBC connect string (`jdbc:oracle:thin:`*`username`*`/`*`password`*`@`*`host`*`:`*`port`*`:`*`sid`*) or a JNDI name for the database.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.1.1.3 sequence-next-val

Returns the next value of an Oracle sequence.

The next value is obtained by executing the following:

```
SELECT sequence.nextval FROM dual
```

against a data source that can be either a JDBC connect string (`jdbc:oracle:thin:username/password@host:port:sid`) or a data source JNDI identifier. Only the Oracle thin driver is supported if a JDBC connect string is used.

Example: `oraext:sequence-next-val('employee_id_sequence','jdbc:oracle:thin:xyz/xyz@localhost:1521:ORCL')`

Signature:

```
oraext:sequence-next-val(sequence as string, data source as string)
```

Arguments:

- `sequence` - The sequence number in the database.
- `data source` - Either a JDBC connect string or a data source JNDI identifier.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.1.2 Date Functions

This section describes the following functions:

B.1.2.1 add-dayTimeDuration-to-dateTime

This function returns a new date time value adding `dateTime` to the given duration.

If the duration value is negative, then the resulting value precedes `dateTime`.

Signature:

```
xpath20:add-dayTimeDuration-from-dateTime(dateTime as string, duration as string)
```

Arguments:

- `dateTime as string` - The `dateTime` to which the function adds the duration, in string format.
- `duration as string` - The duration to add to the `dateTime`, or subtract if the duration is negative, in string format.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xpath20`

B.1.2.2 current-date

This function returns the current date in the ISO format of `YYYY-MM-DD`.

Signature:

```
xpath20:current-date(object)
```

Arguments:

- `Object` - The time in standard format.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xpath20`

B.1.2.3 current-dateTime

This function returns the current datetime value in the ISO format of `CCYY-MM-DDThh:mm:ssTZD`.

Signature:

```
xpath20:current-dateTime(object)
```

Arguments:

- `object` - The time in standard format.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xpath20`

B.1.2.4 current-time

This function returns the current time in ISO format. The format is `hh:mm:ssTZD`.

Signature:

```
xpath20:current-time(object)
```

Arguments:

- `object` - The time in standard format.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xpath20`

B.1.2.5 day-from-dateTime

This function returns the day from `dateTime`. The default day is `1`.

Signature:

```
xpath20:day-from-dateTime(object)
```

Arguments:

- `object` - The time in standard format as a string.

Property IDs:

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: xpath20

B.1.2.6 format-dateTime

This function returns the formatted string of dateTime using the format provided.

Signature:

```
xpath20:format-dateTime(dateTime as string, format as string)
```

Arguments:

- dateTime - The dateTime to be formatted.
- format - The format for the output.

Property IDs:

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: xpath20

B.1.2.7 hours-from-dateTime

This function returns the hour from dateTime. The default hour is 0.

Signature:

```
xpath20:hours-from-dateTime(dateTime as string)
```

Arguments:

- dateTime - The string with the date and time.

Property IDs:

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: xpath20

B.1.2.8 implicit-timezone

This function returns the current time zone in the ISO format of +/- hh:mm, indicating a deviation from Coordinated Universal Timezone (UTC).

Signature:

```
xpath20:implicit-timezone(object)
```

Arguments:

- object - The time in standard format.

Property IDs:

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: xpath20

B.1.2.9 minutes-from-dateTime

This function returns the minute from `dateTime`. The default minute is `0`.

Signature:

```
xpath20:minutes-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime as string` - The date and time.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xpath20`

B.1.2.10 month-from-dateTime

This function returns the month from `dateTime`. The default month is `1` (January).

Signature:

```
xpath20:month-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime as string` - The `dateTime` to be formatted.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xpath20`

B.1.2.11 seconds-from-dateTime

This function returns the second from `dateTime`. The default second is `0`.

Signature:

```
xpath20:seconds-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime as a string` - The `dateTime` as a string.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xpath20`

B.1.2.12 subtract-dayTimeDuration-from-dateTime

This function returns a new `dateTime` value after subtracting the duration from `dateTime`.

If the duration value is negative, then the resulting `dateTime` value follows `input-dateTime` value.

Signature:

```
xpath20:subtract-dayTimeDuration-from-dateTime(dateTime as
string, duration as string)
```

Arguments:

- `dateTime as string` - The `dateTime` from which the function subtracts the duration, in string format.
- `duration as string` - The duration to subtract from the `dateTime`, or to add if the duration is negative, in string format.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

B.1.2.13 timezone-from-dateTime

This function returns the time zone from `dateTime`. The default time zone is `GMT+00:00`.

Signature:

```
xpath20:timezone-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime as string` - The `dateTime` for which this function returns a time zone.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xpath20`

B.1.2.14 year-from-dateTime

This function returns the year from `dateTime`.

Signature:

```
xpath20:year-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime` - The `dateTime` as a string.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xpath20`

B.1.3 Mathematical Functions

This section describes the following function.

B.1.3.1 abs

This function returns the absolute value of `inputNumber`.

If the `inputNumber` is not negative, the `inputNumber` is returned. If the `inputNumber` is negative, the negation of `inputNumber` is returned.

Example: `abs(-1)` returns `1`.

Signature:

```
xpath20:abs(inputNumber as number)
```

Arguments:

- `inputNumber as number` - The number for which the function returns an absolute value.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xpath20`

B.1.4 String Functions

This section describes the string functions.

B.1.4.1 compare

This function returns the lexicographical difference between `inputString` and `compareString` by comparing the unicode value of each character of both the strings.

This function returns `-1` if `inputString` lexicographically precedes the `compareString`.

This function returns `0` if both `inputString` and `compareString` are equal.

This function returns `1` if `inputString` lexicographically follows the `compareString`.

Example: `xpath20:compare('Audi', 'BMW')` returns `-1`

Signature:

```
xpath20:compare(inputString as string, compareString as string)
```

Arguments:

- `variableName` - The source variable for the data.
- `propertyName` - The qualified name (QName) of the property.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xpath20`

B.1.4.2 compare-ignore-case

This function returns the lexicographical difference between `inputString` and `compareString` while ignoring case and comparing the unicode value of each character of both the strings.

This function returns `-1` if `inputString` lexicographically precedes the `compareString`.

This function returns `0` if both `inputString` and `compareString` are equal.

This function returns `1` if `inputString` lexicographically follows the `compareString`.

Example: `xpath20:compare-ignore-case('Audi','bmw')` returns `-1`

Signature:

```
xp:compare-ignore-case(inputString as string, compareString as
string)
```

Arguments:

- `inputString` - The string of data to be searched.
- `CompareString` - The string to compare against the input string.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xpath20`

B.1.4.3 create-delimited-string

This function returns a delimited string created from `nodeSet` delimited by a delimiter.

Signature:

```
oraext:create-delimited-string(nodeSet as node-set, delimiter as
string)
```

Arguments:

- `nodeSet` - The node set to convert into a delimited string.
- `delimiter` - The character that separates the items in the output string; for example, a comma or a semicolon.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

B.1.4.4 ends-with

This function returns `true` if `inputString` ends with `searchString`.

Example: `xpath20:ends-with('XSL Map','Map')` returns `true`

Signature:

```
xpath20:ends-with(inputString as string, searchString as string)
```

Arguments:

- `inputString` - The string of data to be searched.
- `searchString` - The string for which the function searches.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xpath20`

B.1.4.5 format-string

This function returns the message formatted with the arguments passed. At least one argument is required and supports up to a maximum of `10` arguments.

Example: `oraext:format-string('{0} + {1} = {2}','2','2','4')` returns `'2 + 2 = 4'`

Signature:

```
oraext:format-string(string,string,string...)
```

Arguments:

- `string` - One of the strings to be used in the formatted output.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.1.4.6 get-content-as-string

This function returns the XML representation of the input element.

Signature:

```
oraext:get-content-as-string(element as node-set)
```

Arguments:

- `element as node-set` - The input element that the function returns as an XML representation.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.1.4.7 get-content-from-file-function

This function returns the content of the file.

Signature:

```
oraext:get-content-from-file-function(object)
```

Arguments:

- object: The object.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.1.4.8 get-localized-string

This function returns the locale-specific string for the key. This function uses language, country, variant, and resource bundle to identify the correct resource bundle. All parameters must be in string format. Use the `string()` function to convert any parameter values to strings before sending them to `get-localized-string`.

The resource bundle is obtained by resolving `resourceLocation` against the `resourceBaseURL`. The URL is assumed to be a directory only if it ends with `/`.

Usage: `oraext:get-localized-string(resourceBaseURL as string, resourceLocation as string, resource bundle as string, language as string, country as string, variant as string, key as string)`

Example:
`oraext:get-localized-string('file:/c:/','','MyResourceBundle','en','US','','MSG_KEY')` returns a locale-specific string from a resource bundle `'MyResourceBundle'` in the `C:\` directory

Signature:

`oraext:get-localized-string(resourceURL,resourceLocation,resourceBundleName,language,country,variant,messageKey)`

Arguments:

- `resourceURL` - The URL of the resource.
- `resourceLocation` - The subdirectory location of the resource.
- `resourceBundleName` - The name of the ZIP file containing the resource bundle.
- `language` - The language of the localized output.
- `country` - The country of the localized output.
- `variant` - The language variant of the localized output.
- `messageKey` - The message key in the resource bundle.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.1.4.9 index-within-string

This function returns the zero-based index of the first occurrence of `searchString` within the `inputString`.

This function returns `-1` if `searchString` is not found.

Example: `oraext:index-within-string('ABCABC, 'B')` returns `1`

Signature:

```
oraext:index-within-string(inputString as string, searchString
as string)
```

Arguments:

- `inputString` - The string of data to be searched.
- searchString - The string for which the function searches in `inputString`.

Property IDs:

- `namespace-uri:`
  `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.servic`
  `es.functions.ExtFunc`
- `namespace-prefix`: `oraext`

B.1.4.10 last-index-within-string

This function returns the zero-based index of the last occurrence of `searchString` within `inputString`.

This function returns `-1` if `searchString` is not found.

Example: `oraext:last-index-within-string('ABCABC', 'B')` returns `4`

Signature:

```
oraext:last-index-within-string(inputString as string,
searchString as string)
```

Arguments:

- `inputString` - The string of data to be searched.
- searchString - The string for which the function searches in the `inputString`.

Property IDs:

- `namespace-uri:`
  `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.servic`
  `es.functions.ExtFunc`
- `namespace-prefix`: `oraext`

B.1.4.11 left-trim

This function returns the value of `inputString` after removing all the leading white spaces.

Example: `oraext:left-trim('  account  ') returns 'account  '`

Signature:

```
oraext:left-trim(inputString)
```

Arguments:

- `inputString` - The string to be left-trimmed.

Property IDs:

- `namespace-uri:`
  `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.servic`
  `es.functions.ExtFunc`

- namespace-prefix: oraext

B.1.4.12 lower-case

This function returns the value of inputString after translating every character to its lower-case correspondent.

Example: xpath20:lower-case('ABc!D') returns 'abc!d'

Signature:

```
xpath20:lower-case(inputString)
```

Arguments:

- inputString - The string of data that is in lowercase.

Property IDs:

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: xpath20

B.1.4.13 matches

This function returns true if intputString matches the regular expression pattern regexPattern.

Example: xpath20:matches('abracadabra', '^a.*a$') returns true

Signature:

```
xpath20:matches(intputString, regexPattern)
```

Arguments:

- inputString - The string of data that must be matched.
- regexPattern - The regular expression pattern.

Property IDs:

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: xpath20

B.1.4.14 right-trim

This function returns the value inputString after removing all the trailing white spaces.

Example: oraext:right-trim(' account ') returns ' account'

Signature:

```
oraext:right-trim(inputString as string)
```

Arguments:

- inputString - The input string to be right-trimmed.

Property IDs:

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc
- namespace-prefix: oraext

B.1.4.15 upper-case

This function returns the value of `inputString` after translating every character to its uppercase correspondent.

Example: `xpath20:upper-case('abCd0')` returns `'ABCD0'`

Signature:

```
xpath20:upper-case(inputString as string)
```

Arguments:

- `inputString` - The string of data that is in uppercase.

Property IDs:

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: xpath20

B.2 BPEL XPath Extension Functions

This section describes the following BPEL XPath extension functions.

B.2.1 addQuotes

This function returns the content of a `string` with single quotes added.

Signature:

```
ora:addQuotes(string)
```

Arguments:

- `string` - The string to which this function adds quotes.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.2 authenticate

This function authenticates an LDAP user and returns `true` or `false`.

The `authenticate`, `listUsers`, `lookupUser`, and `search` XPath functions provide the lookup and search functionality to obtain information from the LDAP server (typically, the LDAP user details).

These XPath functions use a configuration file to obtain server access information for the JNDI (for example, context factory, LDAP server provider URL, authenticate type, and so on). The configuration file is named `directories.xml` and must be placed in the same directory in which the `.bpel` file for the BPEL project is located. To call these XPath functions, you must provide this file.

Example B–1 shows the format of the `directories.xml` file:

Example B–1 directories.xml File Format

```
<?xml version="1.0" ?>
<directories>
<directory name='people'>
<property name="java.naming.provider.url">ldap://servername:port</property>
<property
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</property>
<property name="java.naming.security.principal">[username]</property>
<property name="java.naming.security.authentication">simple</property>

<property name="java.naming.security.credentials">[passord]</property>
<property name="entryDN">[entry dn]</property>

</directory>
</directories>
```

Example B–2 shows an example of the `directories.xml` file.

Example B–2 directories.xml File Example

```
<?xml version="1.0" ?>
<directories>
<directory name='people'>
<property
name="java.naming.provider.url">ldap://stadb68.us.oracle.com:7001</property>
<property
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</property>
<property name="java.naming.security.principal">cn=admin</property>
<property name="java.naming.security.credentials">weblogic</property>
<property name="java.naming.security.authentication">simple</property>
<property name="entryDN">ou=people,ou=myrealm,dc=soainfra</property>
</directory>
</directories>
```

- **Signature**:

  `ldap:authenticate('directoryName','userId','password')`

- **Parameters**:
  - `directoryName` - The directory name specified in the `directories.xml` file.
  - `userId` - The LDAP server login user ID.
  - `password` - The LDAP server login password.
- **Return**:

  `true` or `false`

  Example:

  `ldap:authenticate('people','weblogic','weblogic')`

  For this XPath function, only two properties must be specified in the `directories.xml` file:
  - `java.naming.provider.url`
  - `java.naming.factory.initial`

B.2.3 appendToList

> **Note:** The `appendToList` function is deprecated. Oracle recommends that you use the `bpelx:copyList` extension of an `assign` activity to append data to a node list.

This function appends to a node list. The node list with which to append should not be null or empty.

Signature:

```
ora:appendToList('variableName', 'partName'?, 'locationPath'?,
Object)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).
- `locationPath` - Provides an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).
- `Object` - The object can be either a list or a single item. If the object is a list, this function appends each item in the list. Each appended item is either an element, or an element with the string value of the node created.

Property IDs:

- `deprecated`

  Use the `bpelx:copyList` or `bpelx:append` extension activity to append to a list.
- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.4 copyList

> **Note:** While the `copyList` function is still available for use, Oracle recommends that you use the `bpelx:copyList` extension to copy a node list or a node. For more information, see Section 6.14.6, "How to Use bpelx:copyList."

This function copies a node list or a node. The node list to be copied to should not be null or empty.

Signature:

```
ora:copyList('variableName', 'partName'?, 'locationPath'?,
Object)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).

- `locationPath` - Provides an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).
- `Object` - The object can be either a list or a single item. If the object is a list, each item in the list is copied. Each item to be copied is either an element, or an element with the string value of the node created.

Property IDs:

- `deprecated`

  Use the `bpelx:copyList` extension activity to append to a list.
- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.5 countNodes

> **Note:** While the `countNodes` function is still available for use, Oracle recommends that you use version 1.0 of the XPath `count()` function to return the size of the elements as an integer.

This function returns the size of the elements as an integer.

Signature:

```
ora:countNodes('variableName', 'partName'?, 'locationPath'?)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).
- `locationPath` - Provides an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.6 doc

This function returns the content of an XML file.

Signature:

```
ora:doc('fileName','xpath'?)
```

Arguments:

- `fileName` - The name of the XML file.
- `xpath` - A part of an XML file (for example, the node set, node list, or leaf node).

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.7 doStreamingTranslate

This function translates using the streaming XPath APIs. It uses a unique concept called batching so that the transformation engine does not materialize the result of a transformation into memory. Therefore, it can handle arbitrarily large payloads of the order of gigabytes. However, it can only handle forward-only XSL constructs such as `for-each`. The `targetType` can be `SDOM` or `ATTACHMENT`.

Signature:

```
ora:doStreamingTranslate('input SDOM or attachment element',
'streaming xpath context', 'SDOM or ATTACHMENT', 'attachment
element?')
```

Arguments:

- `input SDOM or attachment element`
- `streaming xpath context`
- `SDOM or ATTACHMENT`
- `attachment element`

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.8 doTranslateFromNative

This function translates the input data to XML, where the input can be a string, attachment, or element that contains Base64-encoded data. The `targetType` can be `DOM`, `ATTACHMENT` or `SDOM`.

Signature:

```
ora:doTranslateFromNative('input','nxsdTemplate','nxsdRoot','tar
getType','attachment element?')
```

Arguments:

- `input` - The input data of the XPath function.
- `nxsdTemplate` - The NXSD schema to use to translate the input data to XML format.
- `nxsdRoot` - The root element in the native XSD (NXSD) schema.
- `targetType` - Decides how the XPath function translates the native data into XML.
- `attachment element` - This is the attachment for the returned XML. This parameter is optional.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.9 doTranslateToNative

This function translates the input DOM to a string or attachment. The `targetType` can be `STRING` or `ATTACHMENT`

Signature:

```
ora:doTranslateToNative('input','nxsdTemplate','nxsdRoot','targe
tType','attachment element?')
```

Arguments:

- `input` - The input data of the XPath function. The data can either be DOM or SDOM data that must be translated to a native format such as comma-separated values (CSV).

  The input node is usually the root element of the incoming DOM. Example B–3 provides details.

Example B–3 doTranslateToNative Function

```
med:doTranslateToNative($in.request/inp1:Root-Element, 'xsd/address_csv.xsd',
@ 'Root-Element','STRING')"
```

  However, the input node can be a subelement and not the root element of the incoming DOM. Example B–4 provides details.

Example B–4 doTranslateToNative Function

```
med:doTranslateToNative($in.request/inp1:requestToNative/ns1:Root-Element,
 'xsd/address_csv.xsd', 'Root-Element','ATTACHMENT',
 $in.request/inp1:requestToNative/inp1:attachment)
```

  In these situations, you must set the following property in the schema node of the NXSD for this function to execute properly.

  ```
  nxsd:useArrayIdentifiers="true"
  ```

  Note that this setting can adversely impact the performance of this function for very large inputs (in which case, use the `dostreamingxlate` function).

- `nxsdTemplate` - The NXSD schema to use to translate the input data to XML format.
- `nxsdRoot` - The root element in the NXSD schema.
- `targetType` - Decides how the XPath function translates the native data into XML.
- `attachment element` - This is the attachment for the returned XML. This parameter is optional.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.10 doXSLTransform

This function implements WS-BPEL 2.0's `doXSLTransform` function that supports multiple parameters of XSLT. When using this function, the XSL template match must not be set to root (which is `/`). It must be the root element.

Signature:

```
ora:doXSLTransform('url_to_
xslt',input,['paramQname',paramValue]*)
```

Arguments:

- `url_to_xslt` - Specifies the XSL style sheet URL.
- `input` - Specifies the input variable name.
- `paramQname` - Specifies the parameter QName.
- `paramValue` - Specifies the value of the parameter.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.11 doXSLTransformForDoc

This function is a complement XPath function to `doXSLTransform()`. It aims to perform the transformation when the XSLT template matches the document.

Example B–5 shows the `doXSLTransformForDoc` function.

Example B–5 doXSLTransformForDoc Functions

```
<function name="ora:doXSLTransformForDoc">
  <className>com.collaxa.cube.xml.xpath.functions.xml.DoXSLTransformForDocument
  </className>
  <return type="node-set"/>
  <params>
   <param name="template" type="string"/>
   <param name="input" type="string"/>
   <param name="properties" type="string" minOccurs="0" maxOccurs="unbounded"/>
  </params>
  <desc resourceKey="PI_FUNCTION_DESC_DOXSLTRANSFORM_FOR_DOC"></desc>
  <detail resourceKey="PI_FUNCTION_DESC_LONG_DOXSLTRANSFORM_FOR_DOC">
        This function is a complement xpath function to doXSLTransform(). It aims
 to do the transformation when the xslt template matching the
document. The signature of this function is <i>ora:doXSLTransformForDoc('url_to_
xslt',input,['paramQname',paramValue]*)</i>.
        </detail>
  <group>BPEL XPath Extension Functions</group>
 </function>
```

Signature:

```
ora:doXSLTransformForDoc('url_to_
xslt',input,['paramQname',paramValue]*)
```

Arguments:

- `url_to_xslt` - Specifies the XSL style sheet URL.
- `input` - Specifies the input variable name.
- `paramQname` - Specifies the parameter QName.
- `paramValue` - Specifies the value of the parameter.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

You can use the `ora:doXSLTransformForDoc` function to write the results of large XSLT/XQuery operations to a temporary file in a directory system. The document is then loaded from the temporary file when needed. This eliminates the need for caching an entire document as binary XML in memory.

For more information, see Section 42.1.3.7, "Using XPath Functions to Write Large XSLT/XQuery Output to a File System."

B.2.12 formatDate

This function converts standard XSD date formats to characters suitable for output.

Signature:

```
ora:formatDate('dateTime','format')
```

Arguments:

- `dateTime` - Contains a date-related value in XSD format. For nonstring arguments, this function behaves as if a `string()` function were applied. If the argument is not a date, the output is an empty string. If it is a valid XSD date and some fields are empty, this function attempts to fill unspecified fields. For example, `2003-06-10T15:56:00`.
- `format` - Contains a string formatted according to `java.text.SimpleDateFormat` format.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.13 generateGUID

Generates a unique GUID.

Signature:

```
ora:generateGUID()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.14 getApplicationName

This function returns the application name.

Signature:

```
ora:getApplicationName()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.15 getAttachmentContent

This function gets the attachment content from an `href` function.

Signature:

```
ora:getAttachmentContent(varName[, partName[, query]])
```

Arguments:

- `varName` - Specifies the source variable for the data.
- `partName` - (Optional) Specifies the part to select from the variable.
- `query` - (Optional) Specifies an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.16 getComponentName

This function returns the component name.

Signature:

```
ora:getComponentName()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.17 getComponentInstanceID

This function returns the component instance ID.

Signature:

```
ora:getComponentInstanceID()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.18 getCompositeName

This function returns the composite name.

Signature:

```
ora:getCompositeName()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`

- namespace-prefix: ora

B.2.19 getCompositeInstanceID

This function returns the BPEL process composite instance ID.

Signature:

```
ora:getCompositeInstanceID()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.20 getCompositeURL

This function returns the composite URL.

Signature:

```
ora:getCompositeURL()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.21 getContentAsString

This function returns the content of an element as an XML string.

Signature:

```
ora:getContentAsString(element elementAsNodeList)
```

Arguments:

- element - The element (source of the data).
- elementAsNodeList - The element as the node list.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.22 getConversationId

This function returns the conversation ID.

Signature:

```
ora:getConversationId()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.23 getCreator

This function returns the instance creator.

Signature:

```
ora:getCreator()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.24 getCurrentDate

This function returns the current date as a string.

Signature:

```
ora:getCurrentDate('format'?)
```

Argument:

- `format` - (Optional) Specifies a string formatted according to `java.text.SimpleDateFormat` format (optional).

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

For more information, see Section 6.12.1, "How to Assign a Date or Time."

B.2.25 getCurrentDateTime

This function returns the current date time as a string.

Signature:

```
ora:getCurrentDateTime('format'?)
```

Argument:

- `format` - (Optional) Specifies a string formatted according to `java.text.SimpleDateFormat` format (optional).

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.26 getCurrentTime

This function returns the current time as a string.

Signature:

```
ora:getCurrentTime('format'?)
```

Argument:

- `format` - (Optional) Specifies a string formatted according to `java.text.SimpleDateFormat` format (optional).

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.27 getDomainId

This function returns the current domain ID.

Signature:

```
ora:getDomainId()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.28 getECID

This function returns the execution context ID (ECID).

Signature:

```
ora:getECID()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.29 getElement

This function returns an element using `index` from the array of elements.

Signature:

```
ora:getElement('variableName', 'partName', 'locationPath',
index)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (required).
- `locationPath` - Provides an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (required).
- `index` - Dynamic index value. The index of the first node is `1`.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.30 getFaultAsString

This function returns the fault as a string value.

Signature:

```
ora:getFaultAsString()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.31 getFaultName

This function returns the fault name.

Signature:

```
ora:getFaultName()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.32 getGroupIdsFromGroupAlias

This function returns a `List` of user IDs for a group alias specified in the `TaskServiceAliases` section of the BPEL suitcase descriptor.

Signature:

```
ora:getGroupIdsFromGroupAlias(String aliasName)
```

Arguments:

- `aliasName` - The alias for a list of users or groups.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.33 getInstanceId

This function returns the instance ID.

Signature:

```
ora:getInstanceId()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.34 getNodeValue

This function returns the value of a DOM node as a string.

Signature:

`ora:getNodeValue(node)`

Arguments:

- `node` - The DOM node.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.35 getNodes

This function gets a node list. This is implemented as an alternate to `bpws:getVariableData`, which does not return a node list.

Signature:

`ora:getNodes('variableName', 'partName'?, 'locationPath'?)`

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).
- `locationPath` - Provides an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.36 getOwnerDocument

This function returns the document object associated with the node.

Signature:

`ora:getOwnerDocument(node)`

Arguments:

- `node` - Specifies the XML node.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.37 getParentComponentInstanceID

This function returns the BPEL process instance parent component instance ID.

Signature:

`ora:getParentComponentInstanceID()`

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`

- namespace-prefix: ora

B.2.38 getPreference

This function returns the value of a property specified in the preferences section of the BPEL suitcase descriptor.

Signature:

```
ora:getPreference(preferenceName)
```

Arguments:

- preferenceName - The name of the preference as specified in the BPEL suitcase descriptor.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.39 getProcessId

This function returns the ID of the current BPEL process.

Signature:

```
ora:getProcessId()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.40 getProcessOwnerId

This function returns the ID of the user who owns the process, if specified in the TaskServiceAliases section of the BPEL suitcase descriptor.

Signature:

```
ora:getProcessOwnerId()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.41 getProcessURL

This function returns the root URL of the current BPEL process.

Signature:

```
ora:getProcessURL()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension

- namespace-prefix: ora

B.2.42 getProcessVersion

This function returns the current process version.

Signature:

```
ora:getProcessVersion()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.43 getUserAliasId

This function returns the user ID for an alias specified in the `TaskServiceAliases` section of the BPEL suitcase descriptor.

Signature:

```
ora:getUserAliasId (String aliasName)
```

Arguments:

- `aliasName` - The alias for a list of users or groups.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.44 getUserIdsFromGroupAlias

This function returns a `List` of user IDs for a group alias specified in the `TaskServiceAliases` section of the BPEL suitcase descriptor.

Signature:

```
ora:getUserIdsFromGroupAlias( String aliasName )
```

Arguments:

- aliasName - Alias name of the group.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.45 setCompositeInstanceTitle

This function sets a title to the composite instance that can later be used as one of the criteria in searching the instances. This function returns the same string that is passed as the argument.

Signature:

```
med:setCompositeInstanceTitle(title)
```

Arguments:

- `title` - Specifies the composite instance title. This can be specified as an XPath expression on the message payload.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.46 instanceOf

This function extracts arbitrary values from BPEL variables.

Signature:

```
ora:instanceOf(an_xpath_expression, 'typeQName')
```

Arguments:

- `an_xpath_expression` - An XPath expression that returns an element.
- typeQName - The QName of a globally-declared XSD type.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.47 integer

This function returns the content of the node as an integer.

Signature:

```
ora:integer(node)
```

Arguments:

- `node` - The input node.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.48 listUsers

This function returns a list of LDAP users.

- Signature:

  ```
  ldap:listUsers('directoryName',filter')
  ```

- Parameters:
  - `directoryName` - The directory name specified in the `directories.xml` file. For information about the `directories.xml` file, see Section B.2.2, "authenticate."
  - `filter` - The filter expression to use for the search; this value cannot be `null`.
- Return

  An XML element that contains a list of users.

For this XPath function, all properties must be specified in the `directories.xml` file.

Example:

```
ldap:listUsers('people','ou=people');
```

Example B–6 provides an example of the output:

Example B–6 listUsers Output

```
<users xmlns="http://schemas.oracle.com/bpel/ldap">
   <user dn="uid=weblogic">
      <uid>weblogic</uid>
      <userpassword>
Unknown macro: {ssha}

bHDVJRfWVt/Uwlzb4TKU+QTOLB4FLySO</userpassword>

      <objectclass>inetOrgPerson</objectclass>
      <objectclass>organizationalPerson</objectclass>
      <objectclass>person</objectclass>
      <objectclass>top</objectclass>
      <objectclass>wlsUser</objectclass>
      <description>This user is the default administrator.</description>
      <wlsMemberOf>cn=Administrators,ou=groups,ou=myrealm,dc=soainfra</wlsMember
Of>
      <orclguid>8AC1B6206FDD11DEBF9A7F3D47003274</orclguid>
      <sn>weblogic</sn>
      <cn>weblogic</cn>
   </user>
</users>
```

B.2.49 lookupUser

This function returns LDAP user information.

- Signature:

  ```
  ldap:lookupUser('directoryName','userId')
  ```

- Parameters:
  - `directoryName`: The directory name specified in the `directories.xml` file. For information about the `directories.xml` file, see Section B.2.2, "authenticate."
  - `userId`: The user ID to be searched.
- Return

  An XML element that contains the user information.

  For this XPath function, all properties must be specified in the `directories.xml` file:

  Example:

  ```
  ldap:lookupUser('people','ou=people');
  ```

  Example B–7 provides an example of the output:

Example B–7 lookupUser Output

```
<user dn="" xmlns="http://schemas.oracle.com/bpel/ldap">
<ou>people</ou>
<objectclass>organizationalUnit</objectclass>
<objectclass>top</objectclass>
<orclguid>8ABB9BA06FDD11DEBF9A7F3D47003274</orclguid>
</user>
```

B.2.50 parseEscapedXML

This function parses a string to DOM.

Signature:

`oratext:parseEscapedXML(contentString)`

Arguments:

- `contentString` - The string that this function parses to a DOM.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `oratext`

B.2.51 parseXML

This function parses a string to a DOM element.

Signature:

`oratext:parseXML(contentString)`

Arguments:

- `contentString` - The string that this function parses to a DOM element.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `oratext`

B.2.52 processXQuery

This function returns the result of an XQuery transformation.

Signature:

`ora:processXQuery('template','context'?)`

Arguments:

- `template` - The XSLT template.
- `input` - The input data to be transformed.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.53 processXSLT

This function returns the result of an XSLT transformation using the Oracle XDK XSLT processor.

Example B–8 shows the 11*g* version of `processXSLT`.

Example B–8 11g Version of processXSLT

```
<function name="ora:processXSLT">
 <className>com.collaxa.cube.xml.xpath.functions.xml.GetElementFromXDKXSLTFunction
 </className>
  <return type="node-set"/>
  <params>
   <param name="template" type="string"/>
   <param name="input" type="string"/>
   <param name="properties" type="string" minOccurs="0" maxOccurs="unbounded"/>
  </params>
  <desc resourceKey="PI_FUNCTION_DESC_PROCESSXSLT"></desc>
  <detail resourceKey="PI_FUNCTION_DESC_LONG_PROCESSXSLT">
        This function returns result of XSLT transformation by using Oracle XDK
        XSLT processor.
        </detail>
  <group>BPEL XPath Extension Functions</group>
 </function>
```

Example B–9 shows the 10*g* version of `processXSLT`, which is provided for backward compatiblity.

Example B–9 10g Version of processXSLT

```
<function name="xdk:processXSLT">
 <className>com.collaxa.cube.xml.xpath.functions.xml.GetElementFromXDKXSLTFunction
 </className>
  <return type="node-set"/>
  <params>
   <param name="template" type="string"/>
   <param name="input" type="string"/>
   <param name="properties" type="string" minOccurs="0" maxOccurs="unbounded"/>
  </params>
  <desc resourceKey="PI_FUNCTION_DESC_PROCESSXSLT"></desc>
  <detail resourceKey="PI_FUNCTION_DESC_LONG_PROCESSXSLT">
   This is same as 11g, for backward compatiablity this function uses the
   old 10.1.2 namespace. This function returns the result of XSLT transformation
   by using Oracle XDK XSLT processor.
            </detail>
  <group>BPEL XPath Extension Functions</group>
 </function>
```

Signature:

11*g* version of the signature:

```
ora:processXSLT('template','input','properties'?)
```

10*g* version of the signature:

```
xdk:processXSLT('template','input','properties'?)
```

Arguments:

- `template` - The XSLT template. Both HTTP and file URLs are supported.

- `input` - The input data to be transformed.
- `properties` - The properties that translate to XSL parameters that can be accessed within the XSL map using the construct `<xsl:param name="paramName"/>`. The properties are defined as follows:

  1. Create a `params.xsd` file to define the name-value pair (every property is a name-value pair). For example:

```
<?xml version="1.0" encoding="windows-1252" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://schemas.oracle.com/service/bpel/common"
            targetNamespace="http://schemas.oracle.com/service/bpel/common"
            elementFormDefault="qualified">
  <!-- Root Element for Parameters -->
  <xsd:element name="parameters">
    <xsd:complexType>
      <xsd:sequence>
        <!-- Each Parameter is represented by an "item" node that contains
             one unique name and a string value
        -->
        <xsd:element name="item" minOccurs="1" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:string"/>
              <xsd:element name="value" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

  2. Create a `SetParams.xsl` file to populate the properties. Within the XSLT, the parameters are accessible through their names. For this example, the parameter names are `userName` and `location`, and the values are `jsmith` and `CA`, respectively.

```
<?xml version="1.0" encoding="UTF-8" ?>
<?oracle-xsl-mapper
  <mapSources>
    <source type="XSD">
      <schema location="TestXSLParams.xsd"/>
      <rootElement name="TestXSLParamsProcessRequest"
 namespace="http://xmlns.oracle.com/TestXSLParams"/>
    </source>
  </mapSources>
  <mapTargets>
    <target type="XSD">
      <schema location="params.xsd"/>
      <rootElement name="ArrayOfNameAnyTypePairType"
 namespace="http://schemas.oracle.com/service/bpel/common"/>
    </target>
  </mapTargets>
  <!-- GENERATED BY ORACLE XSL MAPPER 10.1.3.1.0(build 061009.0802) AT [WED
 APR 18 14:35:04 PDT 2007]. -->
?>
<xsl:stylesheet version="1.0"
                xmlns:ns2="http://schemas.oracle.com/service/bpel/common"
xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services
```

```
.functions.Xpath20"


xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
               xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:ehdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.
headers.ESBHeaderFunctions"
               xmlns:ns0="http://www.w3.org/2001/XMLSchema"

xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services
.functions.ExtFunc"

xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
               xmlns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"
               xmlns:ns1="http://xmlns.oracle.com/TestXSLParams"
               exclude-result-prefixes="xsl ns0 ns1 ns2 xp20 bpws ora ehdr
 orcl ids hwf">
  <xsl:template match="/">
    <ns2:parameters>
      <ns2:item>
        <ns2:name>
          <xsl:value-of select="'userName'"/>
        </ns2:name>
        <ns2:value>
          <xsl:value-of select="'jsmith'"/>
        </ns2:value>
      </ns2:item>
      <ns2:item>
        <ns2:name>
          <xsl:value-of select="'location'"/>
        </ns2:name>
        <ns2:value>
          <xsl:value-of select="'CA'"/>
        </ns2:value>
      </ns2:item>
    </ns2:parameters>
  </xsl:template>
</xsl:stylesheet>
```

3. Invoke `SetParams.xsl` from the `.bpel` file. For example:
   - Within assign activity `initializeXSLParameters`, you initialize the parameter variable from the specific BPEL variable whose information you want to access from within the XSLT.
   - Within assign activity `executeXSLT`, you invoke the XSLT with the parameters as the `properties` (third) argument of the function `processXSLT`.

   For example:

```
<process name="TestXSLParams"
 . . .
 . . .
   <sequence name="main">
      <receive name="receiveInput" partnerLink="client"
         portType="client:TestXSLParams" operation="initiate"
         variable="inputVariable" createInstance="yes"/>
      <assign name="initializeXSLParameters">
         <bpelx:annotation>
```

```
            <bpelx:pattern>transformation</bpelx:pattern>
         </bpelx:annotation>
         <copy>
            <from expression="ora:processXSLT ('SetParams.xsl',
               bpws:getVariableData('inputVariable','payload'))"/>
            <to variable="propertiesXMLVar"/>
            </copy>
      </assign>
      <assign name="executeXSLT">
         <bpelx:annotation>
             <bpelx:pattern>transformation</bpelx:pattern>
         </bpelx:annotation>

         <copy>
            <from expression="ora:processXSLT('TestXSLParams.xsl',
               bpws:getVariableData('inputVariable','payload'),
               bpws:getVariableData('propertiesXMLVar'))"/>
             <to variable="outputVariable" part="payload"/>
         </copy>
      </assign>
      <invoke name="callbackClient" partnerLink="client"
         portType="client:TestXSLParamsCallback"
         operation="onResult"
         inputVariable="outputVariable"/>
   </sequence>
</process>
```

4. In BPEL, you use the properties to process the XSLT function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora` (for 11*g*)
- `namespace-prefix`: `xdk` (for 10*g*)

You can use the `ora:processXSLT` function to write the results of large XSLT/XQuery operations to a temporary file in a directory system. The document is then loaded from the temporary file when needed. This eliminates the need for caching an entire document as binary XML in memory.

For more information, see Section 42.1.3.7, "Using XPath Functions to Write Large XSLT/XQuery Output to a File System."

B.2.54 processXSLTAttachment

This function returns the results of XSLT transformation by using the Oracle XDK XSLT processor. This function also supports transformations from and to XML attachments.

Signature:

```
ora:processXSLTAttachment('template','input','href'?,'properties
'?)
```

Example B–10 provides an example of signature use.

Example B–10 Signature

```
<function name="ora:processXSLTAttachment">
<className>com.collaxa.cube.xml.xpath.functions.xml.
 GetElementFromXSLTAttachmentFunction</className>
```

```
<return type="node-set"/>
<params>
 <param name="template" type="string"/>
 <param name="input" type="string"/>
 <param name="href" type="string" minOccurs="0"/>
 <param name="properties" type="string" minOccurs="0" maxOccurs="unbounded"/>
</params>
<desc resourceKey="PI_FUNCTION_DESC_processXSLTAttachment"></desc>
<detail resourceKey="PI_FUNCTION_DESC_LONG_processXSLTAttachment">
```

Arguments:

- `template` - The XSLT template.
- `input` - The input data to be transformed.
- `href` - The location of the actual data.
- `properties` - The properties that translate to XSL parameters that can be accessed within the XSL map using the construct `<xsl:param name="paramName"/>`. See Section B.2.53, "processXSLT" for information on defining this argument.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.55 processXSQL

This function returns the result of the XSQL request.

Signature:

```
ora:processXSQL('template','input','properties'?)
```

Arguments:

- `template` - The XSLT template.
- `input` - The input data to be transformed.
- `properties` - The properties that translate to XSL parameters that can be accessed within the XSL map using the construct `<xsl:param name="paramName"/>`. See Section B.2.53, "processXSLT" for information on defining this argument.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.56 readBinaryFromFile

This function reads data from a file.

Signature:

```
ora:readBinaryFromFile(fileName)
```

Arguments:

- `fileName` - The file name from which to read data.

Property IDs:

- namespace-uri:http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.57 readFile

This function returns the content of the file.

Signature:

```
ora:readFile('fileName','nxsdTemplate'?,'nxsdRoot'?)
```

Arguments:

- fileName - The name of the file. This argument can also be an HTTP URL.

  This function by default reads files relative to the suitcase JAR file for the process. If the file to read is located in a different directory path, you must specify an extra directory slash (/) to indicate that this is an absolute path. For example:

  ```
  ora:readFile('file:///c:/temp/test.doc')
  ```

  If you specify only two directory slashes (//), you receive an error similar to that shown in Example B–11:

Example B–11 Error Message with readFile Function

```
XPath expression failed to execute.
Error while processing xpath expression,
the expression is "ora:readFile("file://c:/temp/test.doc")",
the reason is c. Verify the xpath query.
```

- nxsdTemplate - The NXSD template for the output.
- nxsdRoot -The NXSD root.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

Note: Currently, the readFile function does not support the functionality to access files on a web server that requires authorization. If you tried to access such a file, then you get the following error:

```
java.io.IOException: Server returned HTTP response
code: 401 for URL
```

B.2.58 search

This function returns a list of LDAP entries.

- Signature:

  ```
  ldap:search('directoryName','filter','scope')
  ```

- Parameters:
  - directoryName: The directory name specified in the directories.xml file. For information about the directories.xml file, see Section B.2.2, "authenticate."

  - `filter`: The filter expression to use for the search; this value cannot be `null`.
  - `scope`: The scope of the search. It must be one of the following values: `1`: one level, `2`: subtree, or `0`: named object. This parameter is optional. By default, its value is `2`.

- Return

  An XML element that contains the list of entries.

  For this XPath function, all properties must be specified in the `directories.xml` file.

  Example

```
ldap:search('people','cn=weblogic');
```

  Example B–12 provides an example of the output:

Example B–12 search Output

```
 <searchResult xmlns="http://schemas.oracle.com/bpel/ldap">
   <searchResultEntry dn="uid=weblogic" xmlns="urn:oasis:names:tc:DSML:2:0:core">
      <attr name="uid">
         <value>weblogic</value>
      </attr>
      <attr name="userpassword">
         <value>
Unknown macro: {ssha}

bHDVJRfWVt/Uwlzb4TKU+QTOLB4FLySO</value>

      </attr>

      <attr name="objectclass">
         <value>inetOrgPerson</value>
         <value>organizationalPerson</value>
         <value>person</value>
         <value>top</value>
         <value>wlsUser</value>
      </attr>
      <attr name="description">
         <value>This user is the default administrator.</value>
      </attr>
      <attr name="wlsMemberOf">
         <value>cn=Administrators,ou=groups,ou=myrealm,dc=soainfra</value>
      </attr>
      <attr name="orclguid">
         <value>8AC1B6206FDD11DEBF9A7F3D47003274</value>
      </attr>
      <attr name="sn">
         <value>weblogic</value>
      </attr>
      <attr name="cn">
         <value>weblogic</value>
      </attr>
   </searchResultEntry>
   <searchResultEntry xmlns="urn:oasis:names:tc:DSML:2:0:core"/>
</searchResult>
```

B.2.59 writeBinaryToFile

This function writes the binary bytes of a variable (or part of the variable) to a file of the given file name.

Signature:

```
ora:writeBinaryToFile(varName[, partName[, query]])
```

Arguments:

- `varName` - The name of the variable.
- `partName` - The name of the part in the `messageType` variable.
- `query` - The query string to a child of the root element.

Property IDs:

- `namespace-uri:http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.60 BPEL Extension Functions in BPEL 1.1 and BPEL 2.0

This section describes BPEL extension functions.

Table B–1 lists the BPEL extension functions supported by either version 1.1 or version 2.0 of the BPEL specification. If a function is supported by a specific version, it displays for selection in the **BPEL Extension Functions** list of the Expression Builder dialog in Oracle JDeveloper. Otherwise, it does not appear. BPEL version 1.1 functions use the namespace prefix `bpws`. BPEL version 1.1 functions use the namespace prefix `bpel`.

Table B–1 BPEL Extension Functions Supported in BPEL 1.1 or BPEL 2.0

Function	Supported in BPEL 1.1?	Supported in BPEL 2.0?
`bpws:getLinkStatus`	Yes	No
`bpws:getVariableData`	Yes	No
`getVariableProperty`	Yes	No
`bpel:getVariableProperty`	No	Yes

B.2.60.1 getLinkStatus

This function returns a boolean value indicating the status of the link. If the status of the link is positive, the value is `true`. Otherwise, the value is `false`. This function can only be used in a `join` condition.

The `linkName` argument refers to the name of an incoming link for the activity associated with the `join` condition.

Signature:

```
bpws:getLinkStatus ('linkName')
```

Arguments:

- `variableName` - The source variable for the data.
- `propertyName` - The QName of the property.

Property IDs:

- `namespace-uri: http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix: bpws`

B.2.60.2 getVariableData

This function extracts arbitrary values from BPEL variables.

When only the first argument is present, the function extracts the value of the variable, which in this case must be defined using an XML schema simple type or element. Otherwise, the return value of this function is a node set containing the single node representing either an entire part of a message type (if the second argument is present and the third argument is absent) or the result of the selection based on the `locationPath` (if both optional arguments are present).

Signature:

```
bpws:getVariableData ('variableName', 'partName'?,
'locationPath'?)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).
- `locationPath` - Provides an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).

Property IDs:

- `namespace-uri: http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix: bpws`

B.2.60.2.1 selectionFailure Fault is Thrown if the Result Node Set is a Size Other Than One During Execution

According to the *Business Process Execution Language for Web Services Specification*, if the `locationPath` argument selects a node set of a size other than one during execution, the standard fault `bpws:selectionFailure` *must* be thrown by a compliant implementation.

For example, the `count()` function shown in Example B–13 does not work if there are multiple entries of `product` elements under `StoreRequest`; this causes a `selectionFailure` fault to be thrown.

Example B–13 count() Function Error

```
count(bpws:getVariableData('inputVariable',
 'payload','/ns2:StoreRequest/ns2:product'))
```

To make this work, change the syntax to the following:

```
"count($inputVariable.payload/ns2:product)"
```

B.2.60.3 getVariableProperty (For BPEL 1.1)

This function extracts arbitrary values from BPEL variables. The first argument specifies the source variable for the data and the second argument identifies the QName of the property to select from that variable. If the given property selects a node

set of a size other than one during execution, the standard fault `bpws:selectionFailure` is thrown.

Signature:

```
bpws:getVariableProperty ('variableName', 'propertyname')
```

Arguments:

- `variableName` - The source variable for the data.
- `propertyName` - The QName of the property.

Property IDs:

- `namespace-uri`: `http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix`: `bpws`

B.2.60.4 getVariableProperty (For BPEL 2.0)

This function extracts arbitrary values from BPEL variables. The first argument specifies the source variable for the data and the second argument identifies the QName of the property to select from that variable. If the given property selects a node set of a size other than one during execution, the standard fault `bpws:selectionFailure` is thrown.

Signature:

```
bpel:getVariableProperty ('variableName', 'propertyname')
```

Arguments:

- `variableName` - The source variable for the data.
- `propertyName` - The QName of the property. If the given property selects a node set of a size other than one during execution, the standard fault `selectionFailure` is thrown.

Property IDs:

- `namespace-uri`: `http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix`: `bpel`

B.2.61 Utility Functions

This section describes the utility functions.

B.2.61.1 batchProcessActive

This function returns the number of active processes in the batch.

Signature:

```
ora:batchProcessActive(String batchId, String processId)
```

Arguments:

- `batchId` - The ID of the batch.
- `processId` - The ID of the process.

Property IDs:

- `namespace-uri:http://schemas.oracle.com/xpath/extension`

- namespace-prefix: ora

B.2.61.2 batchProcessCompleted

This function returns the number of completed processes in the batch.

Signature:

```
ora:batchProcessCompleted(String batchId, String processId)
```

Arguments:

- batchId - The ID of the batch.
- processId - The ID of the process.

Property IDs:

- namespace-uri:http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.61.3 format

This function formats a message using Java's message format.

Signature:

```
ora:format(formatStrings, args+)
```

Arguments:

- formatStrings - The string of data to be formatted.
- args+ - The arguments referenced by the format specifiers in the format string. If there are more arguments than format specifiers, the extra arguments are ignored. The number of arguments is variable and may be zero.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.61.4 genEmptyElem

This function generates a list of empty elements for the given QName.

Signature:

```
ora:genEmptyElem('ElemQName',size?, 'TypeQName'?, xsiNil?)
```

Arguments:

- ElemQName - The first argument is the QName of the empty elements.
- size - The second optional integer argument for the number of empty elements. If missing, the default size is 1.
- TypeQName - The third optional argument is the QName, which is the xsi:type of the generated empty name. This xsi:type pattern matches SOAPENC:Array. If missing or an empty string, the xsi:type attribute is *not* generated.
- xsiNil - The fourth optional boolean argument is to specify whether the generated empty elements are XSI – nil, provided the element is XSD-nillable. The default is false. If missing or false, xsi:nil is *not* generated.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.61.5 getChildElement

This function gets a child element for the given element.

Signature:

```
ora:getChildElement(element, index)
```

Arguments:

- element - The source for the data.
- index - The integer value of the child element index.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.61.6 getMessage

This function gets a message based on the arguments.

Signature:

```
ora:getMessage(locale, relativeLocation, resourceName,
resourceKey, resourceLocation?)
```

Arguments:

- locale - The locale of the message.
- relativeLocation - The subdirectory or message.
- resourceName - The name of the message resource.
- resourceKey - The key of the resource.
- resourceLocation - The location of the resource.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.61.7 max-value-among-nodeset

This function returns the maximum value from a list of input numbers, the node set inputNumber.

The node set inputNumber can be a collection of text nodes or elements containing text nodes.

In the case of elements, the first text node's value is considered.

Signature:

```
oraext:max-value-among-nodeset(inputNumber as node-set)
```

Arguments:

- inputNumber - The node set of input numbers.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.2.61.8 min-value-among-nodeset

This function returns the minimum value from a list of input numbers, the node set `inputNumbers`.

The node set can be a collection of text nodes or elements containing text nodes.

In the case of elements, the first text node's value is considered.

Signature:

```
oraext:min-value-among-nodeset(inputNumbers as node-set)
```

Arguments:

- `inputNumber` - The node set of input numbers.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.2.61.9 square-root

This function returns the square root of `inputNumber`.

Example: `oraext:square-root(25)` returns `5`

Signature:

```
oraext:square-root(inputNumber as number)
```

Arguments:

- `inputNumber` - The input number for which the function calculates the square root.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.2.61.10 translateFromNative

This function translates the input stream to an XML file.

Signature:

```
ora:translateFromNative('string','nxsdTemplate'?,'nxsdRoot'?)
```

Arguments:

- `string` - The data to convert into an XML file.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` - The root element defined in the XSD file.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.61.11 translateToNative

Translates the XML to the native data.

Signature:

```
ora:translateFromNative('string','nxsdTemplate'?,'nxsdRoot'?)
```

Arguments:

- `string` - The XML file to convert into a string.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` -The root element defined in the XSD file.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: ora

B.2.61.12 translateFromNativeAttachment

This function translates the input stream to XML.

Signature:

```
ora:translateFromNativeAttachment('string','nxsdTemplate'?,'nxsR
oot'?)
```

Arguments:

- `string` - The data to convert into an XML file.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` - The root element defined in the XSD file.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.61.13 translateToNativeAttachment

This function translates XML to the native data.

Signature:

```
ora:translateFromNativeAttachment('string','nxsdTemplate'?,'nxsR
oot'?)
```

Arguments:

- `string` - The data to convert into an XML file.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` - The root element defined in the XSD file.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`

- namespace-prefix: ora

B.3 Oracle Mediator XPath Extension Functions

This section describes the Oracle Mediator XPath extension functions.

B.3.1 doStreamingTranslate

This function translates using the streaming XPath APIs. It uses a unique concept called batching so that the transformation engine does not materialize the result of transformation into memory. Therefore, it can handle arbitrarily large payloads of the order of gigabytes. However, it can only handle forward-only XSL constructs such as `for-each`. The `targetType` can be `SDOM` or `ATTACHMENT`.

Signature:

```
med:doStreamingTranslate('input','streaming xpath
context','targetType','attachment element'?)
```

Arguments:

- `input` - The input data of the XPath function. This can be an SDOM or attachment element.
- `streaming xpath context`
- `targetType` - Determines how the XPath function translates the native data into XML.
- `attachment element` - The attachment for the returned XML. This parameter is optional.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: med

Example:

```
med.doStreamingTranslate($in.request/inp1:request/inp1:sourceAtt
achmentElement,$in.request/inp1:request/inp1:streamingcontext,
'ATTACHMENT',
$in.request/inp1:request/inp1:targetAttachmentElement)
```

B.3.2 doTranslateFromNative

This function translates the input data to XML, where the input can be a string to translate, a file or FTP adapter attachment, an attachment, or an element that contains Base64-encoded data. The `targetType` can be `DOM`, `ATTACHMENT` or `SDOM`.

Signature:

```
med:doTranslateFromNative('input','nxsdTemplate','nxsdRoot','tar
getType','attachment element'?)
```

Arguments:

- `input` - The input data of the XPath function. The data is in a native format, such as comma-separated values (CSV).
- `nxsdTemplate` - The NXSD schema to use to translate the input data to XML format.

- nxsdRoot - The root element in the NXSD schema.
- targetType - Determines how the XPath function translates the native data into XML.
- attachment element - The attachment for the returned XML. This parameter is optional.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: med

Example:

```
med:doTranslateFromNative(string($in.request/inp1:request/inp1:s
ource),'xsd/address_csv.xsd','Root-Element','DOM')
```

B.3.3 doTranslateToNative

This function translates the input DOM to a string or attachment. The targetType can be STRING or ATTACHMENT.

Signature:

```
med:doTranslateToNative('input','nxsdTemplate','nxsdRoot','targe
tType','attachment element'?)
```

Arguments:

- input - The input data of the XPath function. The data can either be DOM or SDOM data that must be translated to a native format such as comma-separated values (CSV).

  The input node is usually the root element of the incoming DOM, as shown in Example B–14.

Example B–14 Input Node as Root Element in doTranslateToNative Function

```
med:doTranslateToNative($in.request/inp1:Root-Element, 'xsd/address_csv.xsd',
 @ 'Root-Element','STRING')"
```

  However, the input node can also be a subelement and not the root element of the incoming DOM, as shown in Example B–15.

Example B–15 Input Node as Subelement in doTranslateToNative Function

```
med:doTranslateToNative($in.request/inp1:requestToNative/ns1:Root-Element,
 'xsd/address_csv.xsd', 'Root-Element','ATTACHMENT',
 $in.request/inp1:requestToNative/inp1:attachment)
```

  In this case, you must set the useArrayIdenitifer property to true in the schema node of the NXSD, as shown below.

```
nxsd:useArrayIdentifiers="true"
```

  This setting can adversely impact the performance of this function for very large inputs. You can use the dostreamingxlate function in this case.

- nxsdTemplate - The NXSD schema to use to translate the input data to XML format.
- nxsdRoot - The root element in the NXSD schema.

- `targetType` - Determines how the XPath function translates the native data into XML.
- `attachment element` - The attachment for the returned XML. This parameter is optional.

Property IDs::

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `med`

Example:

```
med:doTranslateToNative($in.request/inp1:Root-Element,'xsd/addre
ss_csv.xsd','Root-Element','STRING')
```

B.3.4 getAttachmentContent

This function gets the attachment content and encodes the data into Base64 format.

Signature:

```
med:getAttachmentContent(xpathExpr)
```

Arguments:

- `xpathExpr` - The XPath expression that references the incoming attachment.

Property IDs::

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `med`

Example:

```
med:getAttachmentContent($in.bin/bin)
```

B.3.5 getComponentInstanceID

This function returns the component instance ID.

Signature:

```
mdhr:getComponentInstanceId()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: mdhr`

B.3.6 getComponentName

This function returns the component name.

Signature:

```
mdhr:getComponentName()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`

- namespace-prefix: mdhr

B.3.7 getCompositeInstanceID

This function returns the composite instance ID.

Signature:

```
mdhr:getComponentInstanceId()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: mdhr

B.3.8 getCompositeName

This function returns the composite name.

Signature:

```
mdhr:getCompositeName()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: mdhr

B.3.9 getHeader

This function returns the value of an XPath expression from the Oracle Mediator message header.

> **Note:** The `getHeader` function works only when both parameters are specified.

Signature:

```
mdhr:getHeader(xpath as string, namespaces as string)
```

Arguments:

- `xpath`: Refers to the path you traverse from the schema.
- `namespaces`: Refers to the abstract container that contains the context of the XPath expression. This argument is not optional. Namespace declarations are in the following form:

  ```
  'prefix=namespace;
  ```

  Note the semicolon after the namespace declaration. For example:

  ```
  getHeader("in.header.ns9_name/ns9:name/ns9:first","ns9=http//exmaple.com;")
  ```

  In the XSLT Mapper in Oracle JDeveloper, drag the `getHeader` function into the mapper. In the Edit Function - getHeader dialog, click **Add**. The `namespaces` argument is added for you to enter the required information.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix:mdhr`

B.3.10 getECID

This function returns the ECID.

Signature:

```
mdhr:getECID()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: mdhr`

B.3.11 getParentComponentInstanceID

This function returns the Oracle Mediator instance parent component instance ID.

Signature:

```
mdhr:getParentComponentInstanceId()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: mdhr`

B.3.12 setCompositeInstanceTitle

This function sets a title to the composite instance that can later be used as one of the criteria in searching the instances. This function returns the same string that is passed as the argument.

Signature:

```
mdhr:setCompositeInstanceTitle(title)
```

Arguments:

- `title` - Specifies the composite instance title. This can be specified as an XPath expression on the message payload.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: mdhr`

B.3.13 translateFromNativeAttachment

This function translates the input stream to XML.

Signature:

```
med:translateFromNativeAttachment('input','nxsdTemplate',
'href'?,'nxsdRoot'?)
```

Arguments:

- `input` - The data to convert into an XML file. This can be an attachment or a string.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `href` - The location of the actual data. This is optional.
- `nxsdRoot` - The root element defined in the XSD file. This is optional. If the root is not provided, the root element of the template file specified above is considered; otherwise, the subelement from the template is retrieved.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `med`

Example:

```
med:translateFromNativeAttachment($in.bin/bin, 'xsd/address_
csv.xsd', 'myhref-id', 'Root-Element')
```

B.3.14 translateToNativeAttachment

This function translates XML to the native data.

Signature:

```
med:translateToNativeAttachment('input','nxsdTemplate','href'?,
'nxsdRoot'?)
```

Arguments:

- `input` - The data to convert into an XML file. The input can be an attachment or a string.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `href` - The actual location of the data. This is optional.
- `nxsdRoot` - The root element defined in the XSD file. This is optional. If the root is not provided, the root element of the template file specified above is considered; otherwise, the subelement from the template is retrieved.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `med`

Example:

```
med:translateToNativeAttachment($in.bin/bin,'xsd/address_
csv.xsd','myhref-id','Root-Element')
```

B.4 Advanced Functions

This section describes the advanced functions.

B.4.1 create-nodeset-from-delimited-string

This function takes a delimited string and returns a `nodeSet`.

Signature:

```
oraext:create-nodeset-from-delimited-string(qname,
delimited-string, delimiter)
```

Arguments:

- `qname` - The qualified name in which each node in the node set must be created. The QName can be represented in two forms:
  - `task:assignee`
  - `{http://mytask/task}assignee`
- `delimited-string` - The sting of elements separated by the delimiter.
- `delimiter` - The character that separates the items in the input string; for example, a comma or a semicolon.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.4.2 generate-guid

This function generates a unique GUID.

Signature:

```
oraext:generate-guid()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.4.3 lookupPopulatedColumns

This function looks up a cross-reference column for a single value or multiple values corresponding to a value in a reference column.

Signature:

```
xref:lookupPopulatedColumns(tableName,columnName,value,needAnExc
eption)
```

Arguments:

- `xrefTableName`: The name of the reference table.
- `xrefColumnName`: The name of the reference column.
- `xrefValue`: The value corresponding to the reference column name.
- `needAnException`: If this value is set to `true`, then an exception is thrown when no value is found in the referenced column. Otherwise, an empty node set is returned.

Property IDs:

- `namespace-uri:http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`
- `namespace-prefix: xref`

B.4.4 lookupValue

This function returns a string by looking up the value for the target column in a domain value map, where the source column contains the given source value.

Signature:

```
dvm:lookupValue(dvmLocation,sourceColumnName,sourceValue,targetColumnName,defaultValue)
```

Arguments:

- `dvmLocation`: The domain value map URI.
- `sourceColumnName`: The source column name.
- `sourceValue`: The source value (an XPath expression bound to the source document of the XSLT transformation).
- `targetColumnName`: The target column name.
- `defaultValue`: If the value is not found, then the default value is returned.
- `QualifierSourceColumn`: The name of the qualifier column.
- `QualifierSourceValue`: The value of the qualifier.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue`
- `namespace-prefix: dvm`

For more information, see Section 44.4.1.1, "dvm:lookupValue."

B.4.5 lookupValue1M

This function returns an XML document fragment containing values for multiple target columns of a domain value map, where the value for the source column equals the source value.

Signature:

```
dvm:lookupValue1M(dvmLocation,sourceColumnName,sourceValue,targetColumnName1,targetColumnName2...)
```

Arguments:

- `dvmMetadataURI` - The domain value map URI.
- `SourceColumnName` - The source column name.
- `SourceValue` - The source value (an XPath expression bound to the source document of the XSLT transformation).
- `TargetColumnName` - The name of the target columns. You must specify at least one column name. The question mark symbol (?) indicates that you can specify multiple target column names.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue`
- `namespace-prefix:dvm`

For more information, see Section 44.4.1.2, "dvm:lookupValue1M."

B.4.6 lookupXRef

This function looks up a cross-reference column for a value that corresponds to a value in a reference column.

Signature:

```
xref:lookupXRef(tableName,referenceColumnName,referenceValue,columnName,needAnException)
```

Arguments:

- `xrefLocation`: The cross-reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: When the value is set to `true`, an exception is thrown if the value is not found. Otherwise, an empty value is returned.

Property IDs:

- `namespace-uri:http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`
- `namespace-prefix: xref`

For more information, see Section 46.5.1, "About the xref:lookupXRef Function."

B.4.7 lookupXRef1M

This function looks up a cross-reference column for multiple values corresponding to a value in a reference column.

Signature:

```
xref:lookupXRef1M(tableName,referenceColumnName,referenceValue,columnName,needAnException)
```

Arguments:

- `xrefLocation`: The cross-reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: If this value is set to `true`, then an exception is thrown when the referenced value is not found. Otherwise, an empty node set is returned.

Property IDs:

- `namespace-uri:http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`

- `namespace-prefix: xref`

For more information, see Section 46.5.2, "About the xref:lookupXRef1M Function."

B.4.8 lookup-xml

This function returns the string value of an element defined by `lookupXPath` in an XML file (`docURL`) given its parent XPath (`parentXPath`), the key XPath (`keyXPath`), and the value of the key (`key`).

Example:

`oraext:lookup-xml('file:/d:/country_data.xml', '/Countries/Country', 'Abbreviation', 'FullName', 'UK')` returns the value of the element `FullName` child of `/Countries/Country`, where `Abbreviation = 'UK'` is in the file `D:\country_data.xml`.

Signature:

```
oraext:lookup-xml(docURL, parentXPath, keyXPath, lookupXPath, key)
```

Arguments:

- `docURL` - The XML file.
- `parentXPath` - The parent XPath.
- `keyXPath` - The key XPath.
- `lookupXPath` - The lookup XPath.
- `key` - The key value.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.4.9 markForDelete

This function deletes a value in a cross-reference table. The value in the column is marked as deleted. This function returns `true` if the deletion is successful. Otherwise, it returns `false`.

Signature:

```
xref:markForDelete(tableName,columnName,value)
```

Arguments:

- `xrefTableName`: The cross-reference table name.
- `xrefColumnName`: The name of the column from which you want to delete a value.
- `xrefValueToDelete`: The value to be deleted.

Property IDs:

- `namespace-uri:http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`
- `namespace-prefix: xref`

For more information, see Section 46.6.1, "How to Delete a Cross Reference Table Value."

B.4.10 populateXRefRow

This function populates the column name in the cross-reference table (XREF) in which the reference column has the reference value.

Signature:

```
xref:populateXRefRow(tableName,referenceColumnName,referenceValue,columnName,value,mode)
```

Arguments:

- `xrefLocation`: The cross-reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `xrefvalue`: The value corresponding to the reference column name.
- `xrefmode`: The name of the XREF population mode.

Property IDs:

- `namespace-uri:http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`
- `namespace-prefix: xref`

For more information, see Section 46.4.1, "About the xref:populateXRefRow Function."

B.4.11 populateXRefRow1M

This function populates the column with multiple values in the cross-reference table (XREF) in which the reference column has the reference value.

Signature:

```
xref:populateXRefRow1M(tableName,referenceColumnName,referenceValue,columnName,value,mode)
```

Arguments:

- `xrefLocation`: The cross-reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `xrefvalue`: The value corresponding to the reference column name.
- `xrefmode`: The name of the XREF population mode.

Property IDs:

- `namespace-uri:http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`
- `namespace-prefix: xref`

For more information, see Section 46.4.2, "About the xref:populateXRefRow1M Function."

B.5 Workflow Service Functions

This section describes the workflow service functions.

B.5.1 clearTaskAssignees

This function clears the current task assignees.

Signature:

```
hwf:clearTaskAssignees(taskID)
```

Arguments:

- `task` - The task ID of the task.

Property IDs:

- `namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix: hwf`

B.5.2 createWordMLDocument

This function creates a Microsoft Word ML document as a base 64-encoded string.

Signature:

```
hwf:createWordMLDocument(node, xsltURI)
```

Arguments:

- `node` - The node is an XML node that is an input to the transformation.
- `xsltURI` - The XSLT used to transform the node (the first argument) to Microsoft Word ML.

Property IDs:

- `namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix: hwf`

B.5.3 getNotificationProperty

This function retrieves a notification property. This function evaluates to corresponding values for each notification. Only use this function in the notification content XPath expression. If used elsewhere, it returns `null`.

Signature:

```
hwf:getNotificationProperty(propertyName)
```

Arguments:

- `propertyName` - The name of the notification property. It can be one of the following values:
  - `recipient` - The recipient of the notification.
  - `recipientDisplay` - The display name of the recipient.
  - `taskAssignees` - The task assignees.

  - taskAssigneesDisplay - The display names of the task assignees.
  - locale - The locale of the recipient.
  - taskId - The task ID of the task for which the notification is meant.
  - taskNumber - The task number of the task for which the notification is meant.
  - appLink - The HTML link to the Oracle BPM Worklist task details page.

Property IDs:

- namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
- namespace-prefix: hwf

B.5.4 getNumberOfTaskApprovals

This function computes the number of times the task was approved.

Signature:

```
hwf:getNumberOfTaskApprovals(taskId)
```

Arguments:

- taskId - The ID of the task.

Property IDs:

- namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
- namespace-prefix: hwf

B.5.5 getPreviousTaskApprover

This function retrieves the previous task approver.

Signature:

```
hwf:getPreviousTaskApprover(taskId)
```

Arguments:

- taskId - The ID of the task.

Property IDs:

- namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
- namespace-prefix: hwf

B.5.6 getTaskAttachmentByIndex

This function retrieves the task attachment at the specified index.

Signature:

```
hwf:getTaskAttachmentByIndex(taskId, attachmentIndex)
```

Arguments:

- taskId - The task ID of the task.
- attachmentIndex - The index of the attachment. The index begins at 1. The attachmentIndex argument can be a node whose value evaluates to the index

number as a string (all node values are strings). If specified statically, it can be specified as `'1'`.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.7 getTaskAttachmentByName

This function retrieves the task attachment by the attachment name.

Signature:

```
hwf:getTaskAttachmentByName(taskId, attachmentName)
```

Arguments:

- `taskId` - The task ID of the task.
- `attachmentName` - The name of the attachment.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.8 getTaskAttachmentContents

This function retrieves the task attachment contents by the attachment name.

Signature:

```
hwf:getTaskAttachmentContents(taskId, attachmentName)
```

Arguments:

- `taskId` - The task ID of the task.
- `attachmentName` - The name of the attachment.

Property IDs:

- `namespace-uri`:`http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.9 getTaskAttachmentsCount

This function retrieves the number of task attachments.

Signature:

```
hwf:getTaskAttachmentsCount(taskId)
```

Arguments:

- `taskId` - The task ID of the task.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.10 getTaskResourceBundleString

This function returns the internationalized resource value from the resource bundle associated with a task definition.

Signature:

```
hwf:getTaskResourceBundleString(taskId, key, locale?)
```

Arguments:

- `taskId` - The task ID of the task.
- `key` - The key to the resource.
- `locale` - (Optional) The locale. This value defaults to system locale. This returns a `resourceString` XML element in the namespace `http://xmlns.oracle.com/bpel/services/taskService`, which contains the string from the resource bundle.

Property IDs:

- `namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix: hwf`

B.5.11 wfDynamicGroupAssign

This function gets the name of an identity service group, selected according to the specified assignment pattern. The group is selected from either the subordinate groups of the specified group (if a single group name is supplied), or from the list of groups (if a list of user names is supplied). If the identity service is configured with multiple realms, the realm name for the group and groups must also be supplied. Additional assignment pattern-specific parameters can be supplied. These additional parameters are optional, depending on the details of the specific assignment pattern used.

There are two signatures of this function.

Signature 1:

```
hwf:wfDynamicGroupAssign('patternName','groupName','realmName'?,
'patternParam1'?,'patternParam2'?,...,'patternParamN'?)
```

Argument 1:

- `patternName` - The name of the assignment pattern (for example, `ROUND_ROBIN`).
- `groupName` - The name of the group from which to select a subordinate group.
- `realmName` - The name of the identity service realm to which the group belongs.
- `patternParam1...patternParamN` - Any additional parameters required by the assignment pattern implementation (may be optional, depending on pattern).

Signature 2:

```
hwf:wfDynamicGroupAssign('patternName','groupList','realmName'?,
'patternParam1'?,'patternParam2'?,...,'patternParamN'?)
```

Argument 2:

- `patternName` - The name of the assignment pattern (for example, `ROUND_ROBIN`).
- `groupList` - The list of groups from which to select a group.

- realmName - The name of the identity service realm to which the groups belong.
- patternParam1...patternParamN - Any additional parameters required by the assignment pattern implementation (may be optional, depending on the pattern).

Property IDs:

- namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
- namespace-prefix: hwf

B.5.12 wfDynamicUserAssign

This function returns the name of an identity service user, selected according to the specified assignment pattern. The user is selected from either the subordinate users of the specified group (if a single group name is supplied), or from the list of users (if a list of user names is supplied). If the identity service is configured with multiple realms, the realm name for the group and users must also be supplied. Additional assignment pattern-specific parameters can be supplied. These additional parameters are optional, depending on the details of the specific assignment pattern used.

There are two signatures for this function.

Signature 1:

```
hwf:wfDynamicUserAssign('patternName','groupName','realmName'?,'
patternParam1'?,....,'patternParam2'?,...,'patternParamN'?)
```

Arguments 1:

- patternName - The name of the assignment pattern (for example, ROUND_ROBIN).
- groupName - The name of the group from which to select a subordinate user.
- realmName - The name of the identity service realm to which the group belongs.
- patternParam1 ... patternParamN - Any additional parameters required by the assignment pattern implementation (may be optional, depending on the pattern).

Signature 2:

```
hwf:wfDynamicUserAssign(patternName,userList,realmName?,patternP
aram1?,patternParam2?,...,patternParamN?)
```

Arguments 2:

- patternName - The name of the assignment pattern (for example, ROUND_ROBIN).
- userList - The list of users from which to select a user.
- realmName - The name of the identity service realm to which the users belong.
- patternParam1...patternParamN - Any additional parameters required by the assignment pattern implementation (may be optional, depending on the pattern).

Property IDs:

- namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
- namespace-prefix: hwf

B.5.13 Identity Service Functions

This section describes the identity service functions.

B.5.13.1 getDefaultRealmName

This function returns the default realm name.

Signature:

```
ids:getDefaultRealmName()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix`: `ids`

B.5.13.2 getGroupProperty

This function returns the property value for the given group. If the group or attribute does not exist, it returns `null`.

Signature:

```
ids:getGroupProperty(groupName, attributeName, realmName)
```

Arguments:

- `groupName` - String or element containing the group whose attribute must be retrieved.
- `attributeName` - String or element containing the name of the group attribute. The name is one of the following values:
  1. `displayName`
  2. `email`

  If the identity service uses the LDAP `providerType` or JAZN LDAP-based providers, configure the LDAP server to enable searching by those attributes.
- `realmName` - The realm name. This is optional. If not specified, the default realm is assumed.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix`: `ids`

B.5.13.3 getManager

This function gets the manager of a given user. If the user does not exist or there is no manager for this user, it returns `null`.

Signature:

```
ids:getManager(userName, realmName)
```

Arguments:

- `userName` - The user name.

- `realmName` - The realm name. This is optional. If not specified, the default realm is assumed.

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix: ids`

B.5.13.4 getReportees

This function gets the reportees of the user. If the user does not exist, it returns `null`. This function returns a list of nodes. Each node in the list is called user.

Signature:

```
ids:getReportees(userName, upToLevel, realmName)
```

Arguments:

- `userName` - The user name.
- `upToLevel`- Defines the levels of indirect reportees to be included in the result. If the value is `1`, it returns only direct reportees. If the value is `-1`, it returns all levels of reportees. It can be either an element with value `xsd:number` or a string, for example `'1'`.
- `realmName` - The realm name. This is optional and if not specified, the default realm is assumed.

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix: ids`

B.5.13.5 getSupportedRealmNames

This function returns the supported realm names.

Signature:

```
ids:getSupportedRealms()
```

Property IDs:

- `namespace-uri:`
  `http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix: ids`

B.5.13.6 getUserProperty

This function returns the property of the user. If the user or attribute does not exist, it returns `null`.

Signature:

```
ids:getUserProperty(userName, attributeName, realmName)
```

Arguments:

- `userName` - String or element containing the user whose attribute must be retrieved.

- `attributeName` - The name of the `user` attribute. The attribute name is one of the following values:
  1. `givenName`
  2. `middleName`
  3. `sn`
  4. `displayName`
  5. `mail`
  6. `telephoneNumber`
  7. `homephone`
  8. `mobile`
  9. `facsimile`
  10. `pager`
  11. `preferredlanguage`
  12. `title`
  13. `manager`

  If the identity service uses the LDAP `providerType` or JAZN LDAP-based providers, configure the LDAP server to enable searching by those attributes.
- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix`: `ids`

B.5.13.7 getUserRoles

This function gets the user roles. This function returns a list of objects, either application roles or groups, depending on the `roleType`. If the user or role does not exist, it returns `null`.

Signature:

```
ids:getUserRoles(userName, roleType, direct)
```

Arguments:

- `userName` - String or element containing the user whose roles are to be retrieved.
- `roleType` - The role type that takes one of three values: `ApplicationRole`, `EnterpriseRole`, or `AnyRole`.
- `direct` - A string or element indicating if direct or indirect roles must be fetched. This is optional. If not specified, only direct roles are fetched. This is either `xsd:boolean` or `string true/false`.

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/services/IdentityService`
- `namespace-prefix`: `ids`

B.5.13.8 getusersinapprole

This function returns the list of users who are granted this application role. If either the application role name or the application name provided as input is null, then it returns null.

Signature: `ids:getUsersInAppRole(appRoleName, appName, direct, realmName)`

Arguments:

- appRoleName - String or element containing the application role whose members should be retrieved
- appName - application name within which the application role is created
- direct - String or element indicating if only direct grantees or all users should be fetched
- realmName - String or element containing the realm name. This is optional and, if not specified, the default realm is used.

B.5.13.9 getUsersInGroup

This function gets the users in a group. If the group does not exist, it returns `null`. This function returns a list of nodes. Each node in the list is called `user`.

Signature:

```
ids:getUsersInGroup(groupName, direct, realmName)
```

Arguments:

- `groupName` - The group name.
- `direct` - A boolean flag. If `true`, this function returns direct user grantees; otherwise, all user grantees are returned. It can be either an element with value `xsd:boolean` or string `'true'/'false'`.
- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix`: `ids`

B.5.13.10 isUserInRole

This function verifies if a user has a given role.

Signature:

```
ids:isUserInRole(userID, roleName, realmName)
```

Arguments:

- `userID` - A string or element containing the user whose participation in the role must be verified.
- `roleName` - The role name.
- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix: ids`

B.5.13.11 lookupGroup

This function gets the group. If the group does not exist, it returns `null`.

Signature:

```
ids:lookupGroup(groupName, realmName)
```

Arguments:

- `groupName` - The group name.
- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix: ids`

B.5.13.12 lookupUser

This function gets the user object. If the user does not exist, it returns `null`.

Signature:

```
ids:lookupUser(userName, realmName)
```

Arguments:

- `userName` - The user name.
- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix: ids`

B.6 Building XPath Expressions in Oracle JDeveloper

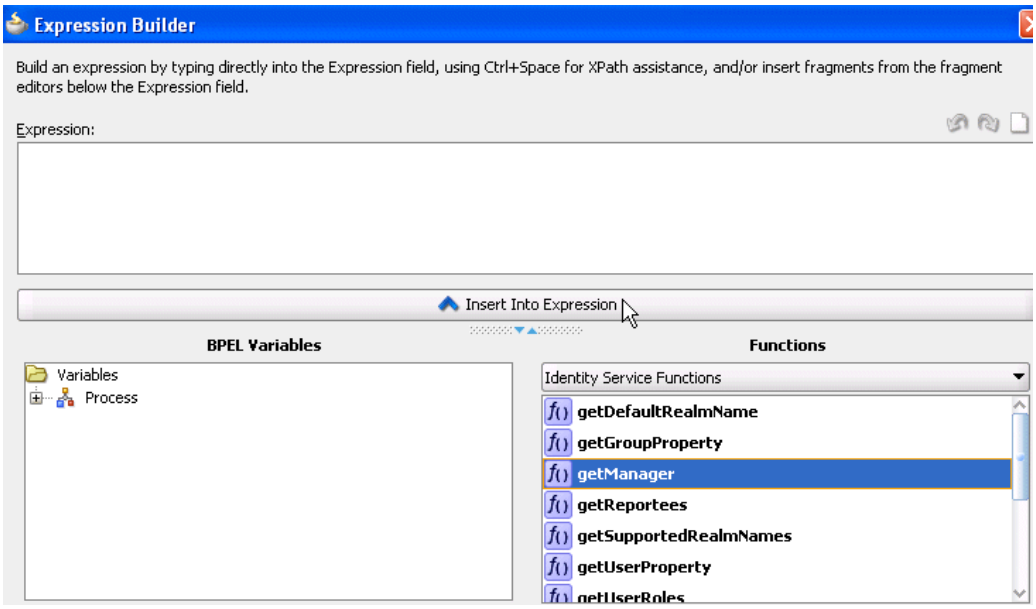
You can use the Expression Builder dialog and the XPath Building Assistant to create XPath expressions. You can visually design XPath expressions in a BPEL process or Oracle Mediator service component in the Expression Builder dialog.

B.6.1 How to Use the Expression Builder

To use the Expression Builder:

1. In the **Functions** list, select the function category to use (for example, **Identity Service Functions**).
2. Select the function (for example, **getManager**).
3. Click **Insert Into Expression**, as shown in Figure B–1.

Figure B–1 Expression Builder Dialog



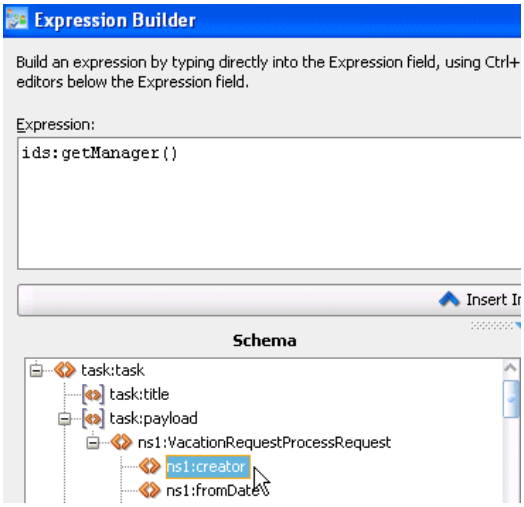
This inserts the function into the **Expression** field at the top.

4. In the **Expression** field, place the cursor between the parentheses of the function, as shown in Figure B–2.

Figure B–2 Placement of Cursor

5. In the **Schema** section, expand the schema path to make your selection, as shown in Figure B–3.

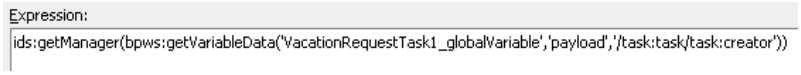
Figure B–3 Selection of Schema



6. Click **Insert Into Expression**.

   The expression is inserted into the function, as shown in Figure B–4.

Figure B–4 XPath Expression Creation



B.6.2 Introduction to the XPath Building Assistant

Several dialogs enable you to specify XPath expressions with the XPath Building Assistant, including:

- **Expression** field of the Expression Builder dialog
- **Expression** field of the **Initialize** tab of the Create Variable dialog in BPEL 2.0
- Edit XPath Expression and Edit Function dialogs of the XSLT Mapper

Manually specifying long and complex expressions is supported, but can be a cumbersome and error-prone process. The XPath Building Assistant provides the following set of features that simplify this process:

- Automatic completion of the following:
  - Elements and attributes
  - Functions
  - BPEL variables and parts
- Function parameter tool tips
- Syntactic and semantic validation of XPaths

B.6.3 How to Use the XPath Building Assistant

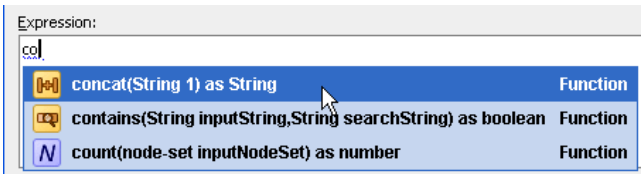
This section provides an example of using the XPath Building Assistant to build an expression in the **Expression** field of the Expression Builder dialog.

To use the XPath Building Assistant:

1. Click inside the **Expression** field and press Ctrl and then the space bar. The menu of available selections is displayed.
2. Make a selection from the list in either of the following ways:
   - Scroll down the list and double-click a function.
   - Enter the beginning letter (for example, c) to display only items starting with that letter and double-click the appropriate function.

   Figure B–5 provides details.

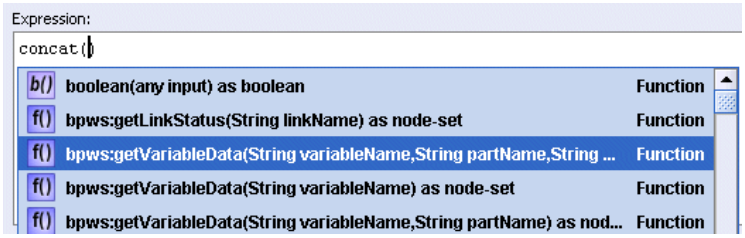
Figure B–5 List of Values for Building an Expression



This value is added to the **Expression** field. The list automatically displays again with different options and prompts you to enter the next portion of the XPath expression.

3. Select and double-click the next portion. Figure B–6 provides details.

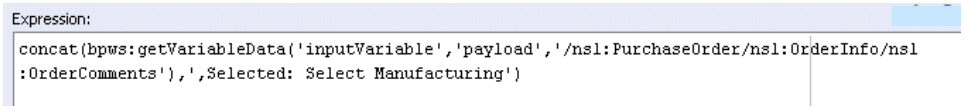
Figure B–6 Invocation of Next Portion of Function



This value is added to the **Expression** field. The list automatically displays again and prompts you to enter the next portion of the XPath expression.

4. Continue this process to build the remaining parts of the XPath expression.
5. Manually add text as appropriate. Figure B–7 provides details.

Figure B–7 Manual Addition of Text



Note: Instead of double-clicking selections in the XPath Building Assistant popups, you can also use the **Enter** key to make the selection. If your expression is complete, but you are still being prompted to enter information, press **Esc**. This closes the list.

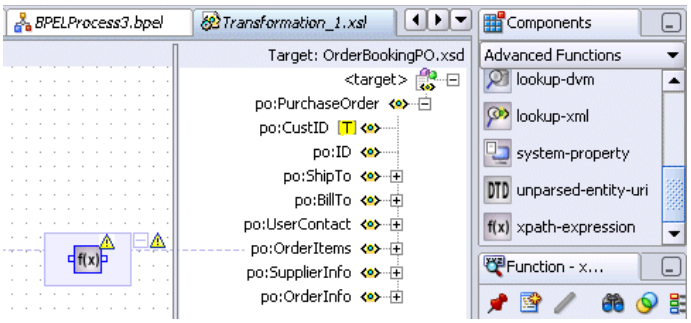
B.6.4 Using the XPath Building Assistant in the XSLT Mapper

This section provides an example of using the XPath Building Assistant to build an expression in the Edit XPath Expression dialog of the XSLT Mapper.

To use the XPath Building Assistant in the XSLT Mapper:

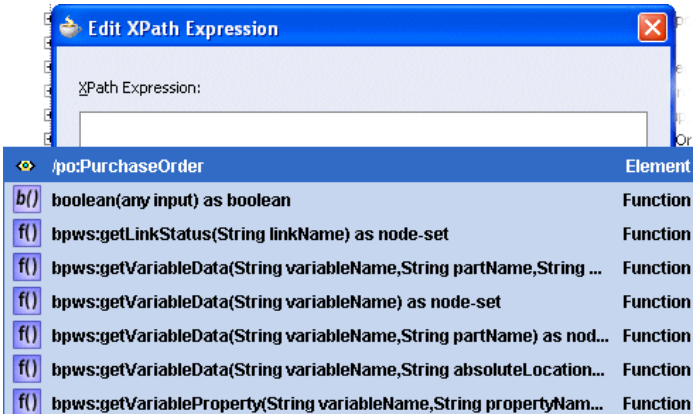
1. Go to the XSLT Mapper.
2. From the **Component Palette** list, select **Advanced Functions**.
3. Scroll down the list to the **xpath-expression** function.
4. Drag and drop the **xpath-expression** function into the XSLT Mapper, as shown in Figure B–8.

Figure B–8 xpath-expression



5. Double-click the function to display the Edit XPath Expression dialog.
6. Click the cursor inside the **XPath Expression** field.
7. Press Ctrl and then the space bar to display a list of values for building an expression, as shown in Figure B–9.

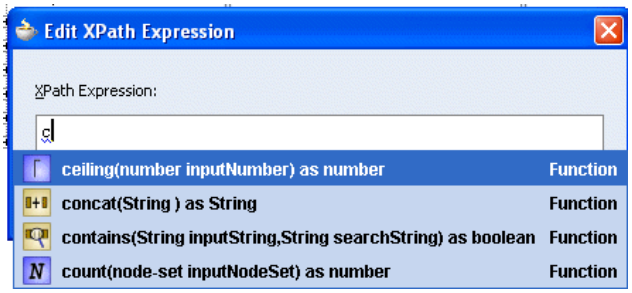
Figure B–9 List of Values for Building an Expression



8. Make a selection from the list (for this example, **concat(String) as String**) in either of the following ways:
   - Scroll down the list and double-click **concat(String) as String**.
   - Enter the letter c to display only items starting with that letter, then select and double-click **concat(String) as String**.

   Figure B–10 provides details.

Figure B–10 Expression List Selection



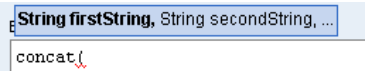
This selection is added to the **XPath Expression** field. The list automatically displays again with different options and prompts you to enter the next portion of the XPath expression.

9. Continue this process to build the remaining parts of the XPath expression.
10. Click **OK** to close the Edit XPath Expression dialog when complete.

B.6.5 Function Parameter Tool Tips

Function parameter tool tips display the expected arguments of a chosen XPath function. For example, if you manually enter the function `concat`, and then enter `(`, the parameter tool tip appears and displays the expected arguments of the `concat` function. The current argument name of the function is highlighted in bold. Figure B–11 provides details.

Figure B–11 Current Argument Name of the Function

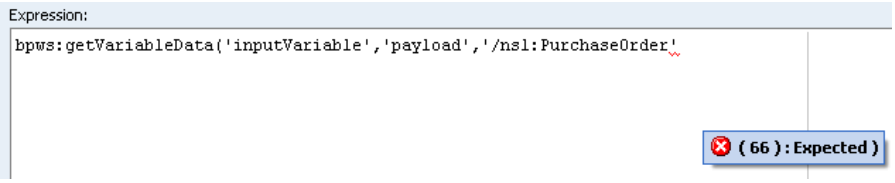


Once you finish specifying one argument, and enter a comma to move to the next argument, the tool tip updates itself to highlight the second argument name in bold, and so on. While editing existing XPaths that contain functions, you can re-invoke parameter tool tips by positioning the cursor within the function and then pressing a combination of the Ctrl, Shift, and space bar keys.

B.6.6 Syntactic and Semantic Validation

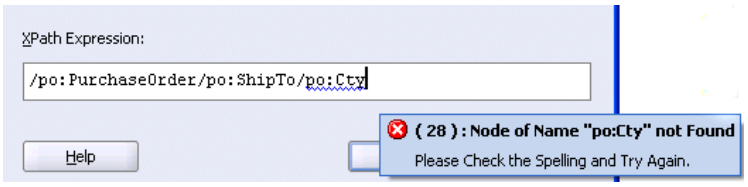
Within Oracle JDeveloper, an XPath expression is considered syntactically valid if it conforms to the XPath 1.0 specification. The XPath Building Assistant warns you about syntactically incorrect XPaths (for example, a missing parenthesis or apostrophe) by underlining the erroneous area in red. Drag the mouse pointer over this area. The error message displays as a tool tip. The red underlining error disappears after you make corrections. Figure B–12 provides details.

Figure B–12 Syntactically Incorrect XPaths



Syntactically valid XPaths may be semantically invalid. This can cause unexpected errors at runtime. For example, you can misspell the name of an element, variable, function, or part. The XPath Building Assistant warns you about semantic errors by underlining the erroneous area in blue. Drag the mouse pointer over this area. The error message displays as a tool tip. The blue underlining error disappears after you make corrections. Figure B–13 provides details.

Figure B–13 Semantically Incorrect XPaths

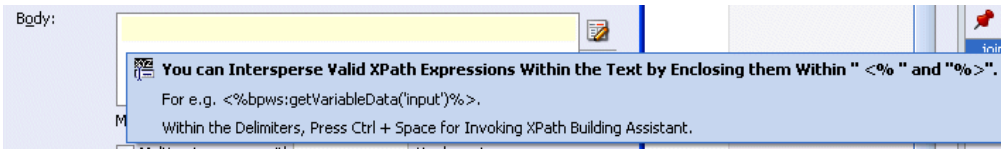


B.6.7 Creating Expressions with Free Form Text and XPath Expressions

You can mix free form text with XPath expressions in some dialogs.

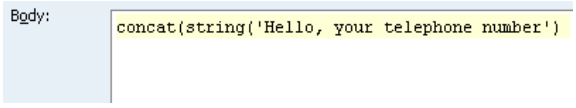
1. Place your cursor over the field to display a popup message that describes this functionality. Figure B–14 provides details.

Figure B–14 Functionality Description Menu



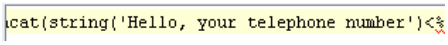
2. Enter free form text (in this example, `'Hello, your telephone number'`). Figure B–15 provides details.

Figure B–15 Free Form Text



3. Enter `<%` when you are ready to invoke the XPath Building Assistant. Figure B–16 provides details.

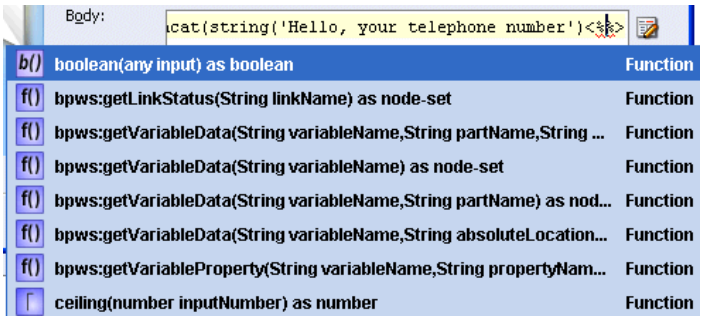
Figure B–16 XPath Building Assistant Invocation Preparation



A red underline appears, which indicates that you are being prompted to add information.

4. Press Ctrl and then the space bar to invoke the XPath Building Assistant. Figure B–17 provides details.

Figure B–17 XPath Building Assistant Invocation



5. Scroll down the list and double-click the value you want.
6. Continue this process to build the remaining parts of the expression.

B.7 Creating User-Defined XPath Extension Functions

You can create user-defined (custom) XPath extension functions for use in Oracle SOA Suite. These functions can be created for the following components:

- Oracle BPEL Process Manager
- Oracle Mediator
- XSLT Mapper
- Human workflow
- Shared by all of these components

XPath extension functions in Oracle SOA Suite adhere to the following standards:

- A single schema defines the configuration syntax for both system functions and user-defined functions.
- XPath functions are categorized based on usage (Oracle BPEL Process Manager, Oracle Mediator, human workflow, XSLT Mapper, and those commonly used by all).
- System functions are separated from user-defined functions.
- A repository hosts both system function configuration files and user-defined function configuration files.
- A repository hosts user-defined function implementation JAR files and automatically makes them available for the Java Virtual Machine (JVM) (class loaders).

As a best practice, follow these conventions for creating functions:

- If possible, write functions that can be shared across all components. Functions shared by all components can be created in a configuration file named `ext-soa-xpath-functions-config.xml`. Note that you must implement XSLT Mapper functions differently than Oracle BPEL Process Manager, Oracle Mediator, and human workflow functions.

  For more information about description of these implementation differences, see Section B.7.1, "How to Implement User-Defined XPath Extension Functions."
- If you create a function for one component that cannot be used by others (for example, a function for Oracle BPEL Process Manager that cannot be used by

Oracle Mediator or human workflow), then create that function in the configuration file specific to that component. For this example, the Oracle BPEL Process Manager function must be created in a configuration file named `ext-bpel-xpath-functions-config.xml`.

Example B–16 shows the function schema used by system and user-defined functions.

Example B–16 Function Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://xmlns.oracle.com/soa/config/xpath"
 targetNamespace="http://xmlns.oracle.com/soa/config/xpath"
 elementFormDefault="qualified">
      <element name="soa-xpath-functions" type="tns:XpathFunctionsConfig"/>
      <element name="function" type="tns:XpathFunction"/>
      <complexType name="XpathFunctionsConfig">
            <sequence>
               <element ref="tns:function" minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="resourceBundle" type="string"/>
            <attribute name="version" type="string"/>
      </complexType>

      <complexType name="XpathFunction">
            <sequence>
                 <element name="className" type="string"/>
                 <element name="return">
                        <complexType>
                               <attribute name="type" type="tns:XpathType"
                                   use="required"/>
                        </complexType>
                 </element>
                 <element name="params" type="tns:Params" minOccurs="0"
                        maxOccurs="1"/>
                 <element name="desc">
                        <complexType>
                               <simpleContent>
                                      <extension base="string">
                                             <attribute name="resourceKey"
                                                 type="string"/>
                                      </extension>
                               </simpleContent>
                        </complexType>
                 </element>
                 <element name="detail" minOccurs="0">
                        <complexType>
                               <simpleContent>
                                      <extension base="string">
                                             <attribute name="resourceKey"
                                                  type="string"/>
                                      </extension>
                               </simpleContent>
                        </complexType>
                 </element>
                 <element name="icon" minOccurs="0">
            <complexType>
                    <simpleContent>
                            <extension base="string">
                                  <attribute name="resourceKey"
```

```
                                        type="string"/>
                                </extension>
                        </simpleContent>
                </complexType>
        </element>
                        <element name="helpURL" minOccurs="0">
                <complexType>
                        <simpleContent>
                                <extension base="string">
                                      <attribute name="resourceKey"
                                         type="string"/>
                                </extension>
                        </simpleContent>
                </complexType>
        </element>
        <element name="group" minOccurs="0">
              <complexType>
                        <simpleContent>
                                <extension base="string">
                                     <attribute name="resourceKey" type="string"/>
                                </extension>
                        </simpleContent>
              </complexType>
        </element>
        <element name="wizardClass" type="string" minOccurs="0"/>
</sequence>
<attribute name="name" type="string" use="required"/>
              <attribute name="deprecated" type="boolean" use="optional"/>
</complexType>

      <complexType name="Params">
      <sequence>
              <element name="param" minOccurs="1" maxOccurs="unbounded">
                      <complexType>
                           <attribute name="name" type="string" use="required"/>
                           <attribute name="type" type="tns:XpathType"
                                 use="required"/>
                           <attribute name="minOccurs" type="string"
                                 default="1"/>
                           <attribute name="maxOccurs" type="string"
                                 default="1"/>
                           <attribute name="wizardEnabled" type="boolean"
                                 default="false"/>
                      </complexType>
              </element>
       </sequence>
  </complexType>
  <simpleType name="XpathType">
       <restriction base="string">
               <enumeration value="string"/>
               <enumeration value="boolean"/>
               <enumeration value="number"/>
               <enumeration value="node-set"/>
               <enumeration value="tree"/>
       </restriction>
  </simpleType>
</schema>
```

B.7.1 How to Implement User-Defined XPath Extension Functions

This section describes how to implement user-defined XPath extension functions for Oracle SOA Suite components.

B.7.1.1 How to Implement Functions for the XSLT Mapper

Implementation of user-defined XPath extension functions for the XSLT Mapper is different than for other components:

- Each XSLT Mapper function requires a corresponding public static method from a public static class. The function name and method name must match.
- XSLT Mapper function namespaces must take the form `http://www.oracle.com/XSL/Transform/java/mypackage.MyFunctionClass`, where `mypackage.MyFunctionClass` is the fully qualified class name of the public static class containing the public static methods for the functions.

For additional details about creating a user-defined XPath extension function for the XSLT Mapper, see Section 37.3.4.4, "Importing User-Defined Functions."

B.7.1.2 How to Implement Functions for All Other Components

For Oracle BPEL Process Manager, Oracle Mediator, and human workflow functions, you must implement either the `oracle.fabric.common.xml.xpath.IXPathFunction` interface (defined in the `fabric-runtime.jar` file) or `javax.xml.xpath.XPathFunction`.

To implement functions for all other components:

1. Implement the `oracle.fabric.common.xml.xpath.IXPathFunction` interface for your XPath function. The `IXPathFunction` interface has one method named `call(context, args)`. The signature of this method is as shown in Example B–17:

Example B–17 Implementation of oracle.fabric.common.xml.xpath.IXPathFunction

```
package oracle.fabric.common.xml.xpath;
public interface IXPathFunction
{
   /** Call this function.
   *
   *  @param context The context at the point in the
   *         expression when the function is called.
   *  @param args List of arguments provided during
   *         the call of the function.
   */
   public Object call(IXPathContext context, List args) throws
XPathFunctionException;
}
```

   where:

   - `context` - The context at the point in the expression when the function is called.
   - `args` - The list of arguments provided during the call of the function.

   For the example shown in Example B–18, a function named `getNodeValue(arg1)` is implemented that gets a value of `w3c node`:

Example B–18 Implementation of getNodeValue(arg1) Function

```
package com.collaxa.cube.xml.xpath.dom.functions;
 import oracle.fabric.common.xml.xpath.IXPathFunction;
 import oracle.fabric.common.xml.xpath.IXPathFunction
 . . .

 public class GetNodeValue implements IXPathFunction {
    Object call(IXPathContext context, List args) throws XPathFunctionException
 {
        org.w3c.dom.Node node = (org.w3c.dom.Node) args.get(0);
        return node.getNodeValue()
    }
 }
```

B.7.2 How to Configure User-Defined XPath Extension Functions

This section describes how to configure user-defined XPath extension functions.

To configure user-defined XPath extension functions:

1. Create an XPath extension configuration file in which to define the function. Example B–19 shows a sample configuration file that follows the function schema shown in Example B–16 of Section B.7, "Creating User-Defined XPath Extension Functions." In this example, two functions are created: `mf:myFunction1` and `mf:myFunction2`.

Example B–19 Sample XML Extension Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<soa-xpath-functions resourceBundle="myPackage.myResourceBundle"
 xmlns="http://xmlns.oracle.com/soa/config/xpath"
 xmlns:mf="http://www.my-functions.com">
  <function name="mf:myFunction1">
    <className>myPackage.myFunctionClass1</className>
    <return type="node-set"/>
    <params>
      <param name="p1" type="node-set" wizardEnabled="true"/>
      <param name="p2" type="string"/>
      <param name="p3" type="number" minOccurs="0"/>
      <param name="p4" type="boolean" minOccurs="0"  maxOccurs="3"/>
    </params>
    <desc resourceKey="func1-desc-key">this is my first function</desc>
    <detail resourceKey="func2-long-desc-key">my first function does ... </detail>
    <icon>myPackage/resource/image/myFunction1.png</icon>
    <group resourceKey="func-group-key">My Function Group</group>
    <wizardClass>myPackage.myWizardClass1</wizardClass>
  </function>
  <function name="mf:myFunction2">
    <className>myPackage.myFunctionClass2</className>
    <return type="string"/>
    <params>
      <param name="p1" type="node-set" wizardEnabled="true"/>
      <param name="p2" type="string"/>
      <param name="p3" type="number" minOccurs="0"/>
      <param name="p4" type="boolean" minOccurs="0"  maxOccurs="unbounded"/>
    </params>
    <desc resourceKey="func2-desc-key">this is my second function</desc>
    <detail resourceKey="func2-long-desc-key">my second function does ...</detail>
    <icon>myPackage/resource/image/myFunction2.png</icon>
    <group resourceKey="func-group-key">My Function Group</group>
```

```
    <wizardClass>myPackage.myWizardClass2</wizardClass>
  </function>
</soa-xpath-functions>
```

Table B–2 describes the elements of the configuration file. Each function configuration file uses `soa-xpath-functions` as its root element. The root element has an optional `resourceBundle` attribute. The `resourceBundle` value is the fully qualified class name of the resource bundle class providing NLS support for all function configurations.

Table B–2 Function Schema Elements

Element	Description
`className`	The fully qualified class name of the function implementation class.
`return`	The return type of the function. This can be one of the following types supported by XPath and XSLT: string, number, boolean, node set, and tree.
`params`	The parameters of the function. A function can have no parameters. A parameter has the following attributes: ■ `name`: The name of the parameter. ■ `type`: The type of the parameter. This can be one of the following types supported by XPath and XSLT: string, number, boolean, node set, and tree. ■ `minOccurs`: The minimum occurrences of the parameter. If set to `0`, the parameter is optional. If set to `1`, the parameter is required. The current restriction is that this attribute must only take a value of either `0` or `1` and that optional parameters must be defined after the required parameters. The default value is `1` if this attribute is absent. ■ `maxOccurs`: The maximum occurrences of the parameter. If set to `unbounded`, the parameter can repeat anytime. This can support functions such as XPath 1.0 function `concat()`, which can take unlimited parameters. The current restriction is that no parameters except the last parameter of the function can have `maxOccurs` greater than `1` or `unbounded`. The default value is `1` if this attribute is absent. ■ `wizardEnabled`: Indicates whether to enable a wizard to enter the parameter value. This supports a user interface where the parameter value must be entered. If set to `true`, a wizard launch button is rendered next to the parameter value field. The wizard launch button, when pressed, launches a popup wizard to help the user enter the parameter value. The wizard class must be specified later. The default value is `false` if this attribute is absent, meaning there is no wizard support for the parameter by default.
`desc`	An optional description of the function. If the `resourceKey` is present, the description is retrieved from the resource bundle specified earlier on the root element.
`detail`	An optional longer (detailed) description of the function. If the `resourceKey` is present, the description is retrieved from the resource bundle specified earlier on the root element.
`icon`	An optional icon URL of the function. If the `resourceKey` is present, the icon URL is retrieved from the resource bundle specified earlier on the root element. This is to support a user interface in which the function must be displayed.

Table B–2 (Cont.) Function Schema Elements

Element	Description
`helpURL`	An optional help HTML URL of the function. If the `resourceKey` is present, the help URL is retrieved from the resource bundle specified earlier on the root element. This is to support a user interface in which the function help link must be displayed.
`group`	An optional group name of the function. If the `resourceKey` is present, the group name is retrieved from the resource bundle specified earlier on the root element. This is to support a user interface where functions must be grouped. If no group name is specified, the function falls into a built-in advanced functions group when being grouped in a user interface.
`wizardClass`	The fully qualified class name of the wizard class for all parameters that are wizard-enabled. This is to support a user interface in which parameter values must be entered. This wizard class is invoked by wizard launch buttons to help you enter parameter values. If there is no wizard-enabled parameter, this element must be absent. **Note:** This element is not supported for user-defined functions. Only system functions currently support this feature.

2. Name your user-defined XPath extension configuration file based on the component type with which to use the function. Table B–3 describes the naming conventions to use for user-defined configuration files.

Table B–3 User-Defined Configuration Files

To Use with This Component...	Use This Configuration File Name...
Oracle BPEL Process Manager	`ext-bpel-xpath-functions-config.xml`
Oracle Mediator	`ext-mediator-xpath-functions-config.xml`
XSLT Mapper	`ext-mapper-xpath-functions-config.xml`
Human workflow	`ext-wf-xpath-functions-config.xml`
All components	`ext-soa-xpath-functions-config.xml`

3. Place the configuration file inside a JAR file along with the compiled classes. Within the JAR file, the configuration file must be located in the `META-INF` directory. The JAR file does not need to reside in a specific directory.

> **Note:** The `customXpathFunction` JAR must be added explicitly as it is not part of the SOA composite.

4. In Oracle JDeveloper, go to **Tools** > **Preferences** > **SOA**.
5. Click the **Add** button and select your JAR file.
6. Restart Oracle JDeveloper for the changes to take effect.

   The JAR file is automatically added to the JVM's class path to make it available for use.

B.7.3 How to Deploy User-Defined Functions to Runtime

The `soa/modules/oracle.soa.ext_11.1.1` directory is provided for adding custom JAR files and classes. For information, see Section 13.3, "Adding Custom Classes and JAR Files."

C

Deployment Descriptor Properties

This appendix describes how to define deployment descriptor properties for BPEL process service components.

This appendix includes the following sections:

- Section C.1, "Introduction to Deployment Descriptor Properties"
- Section C.2, "Deprecated 10.1.3 Properties"

> **Note:** You cannot specify deployment descriptor properties at runtime.

C.1 Introduction to Deployment Descriptor Properties

Deployment descriptors are BPEL process service component properties used at runtime by Oracle WebLogic Server, Oracle Enterprise Manager, or both. There are two types of properties:

- Configuration
- Partner link binding

C.1.1 How to Define Deployment Descriptor Properties

You define configuration properties and values in the BPEL process service component section of the `composite.xml` file. Example C–1 shows how to define the `inMemoryOptimization` configuration property.

Example C–1 Configuration Property Definition in composite.xml

```
...
  <component name="myBPELServiceComponent">
 ....
  <property name="bpel.config.inMemoryOptimization">true</property>
</component>
```

Table C–1 lists the `configuration` deployment descriptor properties.

Table C–1 Properties for the configurations Deployment Descriptors

Property Name	Description
`completionPersistPolicy`	This property configures how the instance data is saved. It can only be set at the BPEL service component level. The following values are available: ■ `on` (default): The completed instance is saved normally. ■ `deferred`: The completed instance is saved, but with a different thread and in another transaction. ■ `faulted`: Only the faulted instances are saved. ■ `off`: No instances of this process are saved.
`disableAsserts`	This property, when set to `true`, disables assertions in BPEL 1.1 projects.
`globalTxMaxRetry`	If using outbound adapters in an asynchronous BPEL process, specify the maximum number of retries for a remote fault.
`globalTxRetryInterval`	If using outbound adapters in an asynchronous BPEL process, specify the time interval in milliseconds between retries for a remote fault.
`inMemoryOptimization`	Default value is `false`. This property can only be set to `true` if it does not have dehydration points. Activities like wait, receive, onMessage, and onAlarm create dehydration points in the process. If this property is set to `true`, in-memory optimization is attempted on the instances of this process on `to-spec` queries.
`keepGlobalVariables`	Specify whether the server can keep global variable values in the instance store when the instance completes: ■ `false` (default): Global variable values are deleted when the instance completes. ■ `true`: Global variable values are not deleted.
`oneWayDeliveryPolicy`	This property sets the persistence policy of the process in the delivery layer. The possible values are: ■ `async.persist`: Messages into the system are saved in the delivery store before being picked up by the engine. ■ `async.cache`: Messages into the system are saved in memory before being picked up by the engine. ■ `sync`: The instance-initiating message is not temporarily saved in the delivery layer. The engine uses the save thread to initiate the message.
`sensorActionLocation`	The location of the sensor action XML file. The sensor action XML file configures the action rule for the events.
`sensorLocation`	The location of the sensor XML file. The sensor XML file defines the list of sensors into which events are logged.
`transaction`	This property configures the transaction behavior of the BPEL instance for initiating calls. ■ `requiresNew`: A new transaction is created for the execution, and the existing transaction (if there is one) is suspended. This behavior is true for both request/response (initiating) environments and one-way, initiating environments in which `bpel.config.oneWayDeliveryPolicy` is set to `sync`. ■ `required`: In request/response (initiating) environments, this setting joins a caller's transaction (if there is one) or creates a new transaction (if there is no transaction). In one-way, initiating environments in which `bpel.config.oneWayDeliveryPolicy` is set to `sync`, the invoke message is processed using the same thread in the same transaction. **Note:** This 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.

You define partner link binding properties and values in the BPEL process service component section of the `composite.xml` file. Example C–2 shows how to define the `nonBlockingInvoke` partner link binding property.

Example C–2 Property Definition in composite.xml

```
...
  <component name="myBPELServiceComponent">
 ....
  <property name="bpel.partnerLink.partner_link_name.
   nonBlockingInvoke">false</property>
</component>
```

Table C–2 lists the `partnerLinkBinding` deployment descriptor properties.

Table C–2 Properties for the partnerLinkBinding Deployment Descriptors

Property Name	Description
`idempotent`	An idempotent activity is an activity that can be retried (for example, an assign activity or an invoke activity). The instance is saved after a nonidempotent activity. This property is applicable to both durable and transient processes. ▪ `true` (default): If the server fails, it performs the activity again after restarting. This is because the server does not dehydrate immediately after the invoke and no record exists that the activity executed. ▪ `false`: Activity is dehydrated immediately after execution and recorded in the dehydration store. When idempotent is set to `false`, it provides better failover protection, but may impact performance if the BPEL process accesses the dehydration store frequently.
`nonBlockingInvoke`	Default value is `false`. When this is set to `true`, a separate thread is spawned to do the invocation so that the invoke activity does not block the instance.

`validateXML`	Enables message boundary validation. When set to `true`, the XML message is validated against the XML schema during a receive activity and an invoke activity for this partner link. If the XML message is invalid, then a `bpelx:invalidVariables` runtime fault is thrown. This overrides the domain level `validateXML` property. The following example enables validation for only the `StarLoanService` partner:
	`<partnerLinkBinding name="StarLoanService">`
	`<property name="wsdlLocation">`
	`http://<hostname>:9700/orabpel/default/StarLoan/Sta`
	`rLoan?wsdl</property>`
	`<property name="validateXML">true</property>`
	`</partnerLinkBinding>`

C.1.2 How to Get the Value of a Preference within a BPEL Process

The value of a property can be read by a BPEL process using the XPath extension function `ora:getPreference(myPref)`. This gets the value of `bpel.preference.myPref`.

This function can be used as part of a simple assign statement, used in condition expressions, or used as part of a more complex XPath expression.

C.2 Deprecated 10.1.3 Properties

Table C–3 lists deprecated properties that can no longer be used.

Table C–3 Deprecated Properties

Property	Deployment Descriptor Type	Deprecated for Release...
completionPersistLevel	configurations	11*g* Release 1
defaultInput	configurations	11*g* Release 1
initializeVariables	configurations	11*g* Release 1
loadSchema	configurations	11*g* Release 1
noAlterWSDL	configurations	11*g* Release 1
optimizeVariableCopy	configurations	11*g* Release 1
relaxTypeChecking	configurations	11*g* Release 1
relaxXPathQName	configurations	11*g* Release 1
transaction	configurations	10.1.3.4
SLACompletionTime	configurations	11*g* Release 1
xpathValidation	configurations	11*g* Release 1
user	configurations	11*g* Release 1
pw	configurations	11*g* Release 1
role	configurations	11*g* Release 1
correlation	partnerLinkBinding	11*g* Release 1
contentType	partnerLinkBinding	10.1.3
retryInterval	partnerLinkBinding	Deprecated by the fault policy feature in 10.1.3.3
retryMaxCount	partnerLinkBinding	Deprecated by the fault policy feature in 10.1.3.3
wsdlLocation	partnerLinkBinding	11*g* Release 1
wsdlRuntimeLocation	partnerLinkBinding	11*g* Release 1
wsseHeaders	partnerLinkBinding	11*g* Release 1
wsseUsername	partnerLinkBinding	11*g* Release 1
wssePassword	partnerLinkBinding	11*g* Release 1
preferredPort	partnerLinkBinding	11*g* Release 1
fullWSAddressing	partnerLinkBinding	11*g* Release 1

D

Understanding Sensor Public Views and the Sensor Actions XSD

This appendix describes the available sensor public views and the sensor actions XSD file that you can import into Oracle BPEL Designer.

This appendix includes the following sections:

- Section D.1, "Introduction to Sensor Public Views and the Sensor Actions XSD File"
- Section D.2, "Sensor Public Views"
- Section D.3, "Sensor Actions XSD File"

For more information, see Chapter 17, "Using Oracle BPEL Process Manager Sensors."

D.1 Introduction to Sensor Public Views and the Sensor Actions XSD File

A set of public views is exposed to allow SQL access to sensor values from literally any application interested in the data. In addition, a sample sensor action schema is provided for importing into Oracle BPEL Designer.

D.2 Sensor Public Views

The sensor framework of Oracle BPEL Process Manager provides the functionality to persist sensor values created by processing BPEL instances in a relational schema stored in the dehydration store of Oracle BPEL Process Manager. The data is used to display the sensor values of a process instance in Oracle Enterprise Manager Fusion Middleware Control.

D.2.1 BPM Schema

The database publisher persists the sensor data in a predefined relational schema in the database. The following public views can be used from a client (Oracle Warehouse, portals, and so on) to query the sensor values using SQL.

Note: In Table D–1 through Table D–4, the Indexed or Unique? column provides unique index names and the order of the attributes. For example, *U1,2* means that the attribute is the second one in a unique index named *U1*. *PK* means primary key.

D.2.1.1 BPEL_PROCESS_INSTANCES

Table D–1 provides an overview of all the process instances of Oracle BPEL Process Manager.

Table D–1 BPEL_PROCESS_INSTANCES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
INSTANCE_KEY	NUMBER	--	PK	N	Unique instance ID
APPLICATION_NAME	VARCHAR2	500	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	500	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	500	--	N	User-defined label
COMPONENT_NAME	VARCHAR2	500	--	N	User-defined component name
TITLE	NVARCHAR2	200	--	Y	User-defined title of the BPEL process
STATE	NUMBER	--	--	Y	State of the BPEL process instance
STATE_TEXT	VARCHAR2	21	--	Y	Text presentation of the state attribute
PRIORITY	NUMBER	--	--	Y	User-defined priority of the BPEL process instance
STATUS	NVARCHAR2	200	--	Y	User-defined status of the BPEL process
STAGE	VARCHAR2	100	--	Y	User-defined stage property of a BPEL process
CONVERSATION_ID	VARCHAR2	256	--	Y	User-defined conversation ID of a BPEL process
CREATION_DATE	TIMESTAMP	6	--	N	Creation time stamp of the process instance
MODIFY_DATE	TIMESTAMP	6	--	Y	Time stamp when the process instance was modified
TS_DATE	DATE	--	--	Y	Date portion of `modify_date`
TS_HOUR	NUMBER	--	--	Y	Hour portion of `modify_date`
EVAL_TIME	NUMBER	--	--	Y	Evaluation time of the process instance in milliseconds

D.2.1.2 BPEL_ACTIVITY_SENSOR_VALUES

Table D–2 contains all the activity sensor values of the monitored BPEL processes.

Table D–2 BPEL_ACTIVITY_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
SENSOR_NAME	NVARCHAR2	200	U1,2	N	The name of the sensor that fired
SENSOR_TARGET	NVARCHAR2	512	--	N	The target of the fired sensor
ACTION_NAME	NVARCHAR2	200	U1,3	N	The name of the sensor action
ACTION_FILTER	NVARCHAR2	512	--	Y	The filter of the action

Table D–2 (Cont.) BPEL_ACTIVITY_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
CREATION_DATE	TIMESTAMP	6	--	N	The creation date of the activity sensor value
MODIFY_DATE	TIMESTAMP	6	--	Y	The time stamp of last modification
TS_DATE	DATE	--	--	Y	Date portion of modify_date
TS_HOUR	NUMBER	--	--	Y	Hour portion of modify_date
CRITERIA_ SATISFIED	VARCHAR2	1	--	Y	NULL, Y, or N
ACTIVITY_NAME	NVARCHAR2	200	--	N	The name of the BPEL activity
ACTIVITY_TYPE	VARCHAR2	30	--	N	The type of the BPEL activity
ACTIVITY_ STATE	VARCHAR2	30	--	Y	The state of the BPEL activity
EVAL_POINT	VARCHAR2	30	--	N	The evaluation point of the activity sensor
ERROR_MESSAGE	NCLOB	--	--	Y	An error message
RETRY_COUNT	NUMBER	--	--	Y	The number of retries of the activity
EVAL_TIME	NUMBER	--	--	Y	Evaluation time of the activity in milliseconds
ID	NUMBER	--	PK	N	Unique ID
INSTANCE_KEY	NUMBER	--	U1,1	N	BPEL process ID
APPLICATION_ NAME	VARCHAR2	500	--	N	User-defined application name
COMPOSITE_ NAME	VARCHAR2	500	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	500	--	N	User-defined label
COMPONENT_ NAME	VARCHAR2	500	--	N	User-defined component name

D.2.1.3 BPEL_FAULT_SENSOR_VALUES

Table D–3 contains all the fault sensor values.

Table D–3 BPEL_FAULT_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ID	NUMBER	--	PK	N	Unique ID
INSTANCE_KEY	NUMBER	--	U1,1	N	BPEL process ID
APPLICATION_ NAME	VARCHAR2	500	--	N	User-defined application name
COMPOSITE_ NAME	VARCHAR2	500	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	500	--	N	User-defined label

Table D–3 (Cont.) BPEL_FAULT_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
COMPONENT_NAME	VARCHAR2	500	--	N	User-defined component name
SENSOR_NAME	NVARCHAR2	200	U1,2	N	The name of the sensor that fired
SENSOR_TARGET	NVARCHAR2	512	--	N	The target of the fired sensor
ACTION_NAME	NVARCHAR2	200	U1,3	N	The name of the sensor action
ACTION_FILTER	NVARCHAR2	512	--	Y	The filter of the action
CREATION_DATE	TIMESTAMP	6	--	N	The creation date of the activity sensor value
MODIFY_DATE	TIMESTAMP	6	--	Y	The time stamp of last modification
TS_DATE	DATE	--	--	Y	Date portion of `modify_date`
TS_HOUR	NUMBER	--	--	Y	Hour portion of `modify_date`
CRITERIA_SATISFIED	VARCHAR2	1	--	Y	`NULL` if no action filter specified; `Y` if action filter is specified and evaluates to true; `N` otherwise
ACTIVITY_NAME	NVARCHAR2	200	--	N	The name of the BPEL activity
ACTIVITY_TYPE	VARCHAR2	30	--	N	The type of the BPEL activity
MESSAGE	CLOB	--	--	Y	The fault message

D.2.1.4 BPEL_VARIABLE_SENSOR_VALUES

Table D–4 contains all the variable sensor values.

Table D–4 BPEL_VARIABLE_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ID	NUMBER	--	PK	N	Unique ID
INSTANCE_KEY	NUMBER	--	U1,1	N	BPEL process ID
APPLICATION_NAME	VARCHAR2	500	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	500	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	500	--	N	User-defined label
COMPONENT_NAME	VARCHAR2	500	--	N	User-defined component name
SENSOR_NAME	NVARCHAR2	200	U1,2	N	Name of the sensor that fired
SENSOR_TARGET	NVARCHAR2	512	--	N	Target of the sensor
ACTION_NAME	NVARCHAR2	200	U1,3	N	Name of the action
ACTION_FILTER	NVARCHAR2	512	--	Y	Filter of the action
ACTIVITY_SENSOR	NUMBER	--	--	Y	ID of the corresponding activity sensor value
CREATION_DATE	TIMESTAMP	6	--	N	Creation date

Table D–4 (Cont.) BPEL_VARIABLE_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
TS_DATE	DATE	--	--	N	Date portion of creation_date
TS_HOUR	NUMBER	--	--	N	Hour portion of creation_date
VARIABLE_NAME	NVARCHAR2	512	--	N	The name of the BPEL variable
EVAL_POINT	VARCHAR2	30	--	Y	Evaluation point of the corresponding activity sensor
CRITERIA_ SATISFIED	VARCHAR2	1	--	Y	NULL, Y, or N
TARGET	NVARCHAR2	512	--	--	--
UPDATER_NAME	NVARCHAR2	200	--	N	The name of the activity or event that updated the variable
UPDATER_TYPE	NVARCHAR2	200	--	N	The type of the BPEL activity or event
SCHEMA_ NAMESPACE	NVARCHAR2	512	--	Y	Namespace of variable sensor value
SCHEMA_ DATATYPE	NVARCHAR2	512	--	Y	Data type of the variable sensor value
VALUE_TYPE	NUMBER	--	--	N	The value type of the variable (corresponds to java.sql.Types values)
VARCHAR2_ VALUE	NVARCHAR2	4000	--	Y	The value of string-like variables
NUMBER_VALUE	NUMBER	--	--	Y	
DATE_VALUE	TIMESTAMP	6	--	Y	User-defined date
DATE_VALUE_TZ	VARCHAR2	10	--	Y	User-defined time zone
BLOB_VALUE	BLOB	--	--	Y	
CLOB_VALUE	CLOB	--	--	Y	

D.3 Sensor Actions XSD File

Example D–1 provides a sample sensor action schema that you can import into Oracle BPEL Designer. This schema is also relevant to custom data publishers.

Example D–1 Sample Sensor Action Schema

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  This schema contains the sensor definition. Sensors monitor data
  and execute callbacks appropriately.
-->
<xsd:schema blockDefault="#all" elementFormDefault="qualified"
            targetNamespace="http://xmlns.oracle.com/bpel/sensor"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:tns="http://xmlns.oracle.com/bpel/sensor">

  <xsd:simpleType name="tSensorActionPublishType">
    <xsd:annotation>
      <xsd:documentation>
        This enumeration lists the possibe publishing types for probes.
```

```
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="BpelReportsSchema"/>
      <xsd:enumeration value="JMSQueue"/>
      <xsd:enumeration value="JMSTopic"/>
      <xsd:enumeration value="Custom"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="tSensorActionProperty">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="name" use="required" type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <!--
    Attributes of a sensor action
  -->
  <xsd:attributeGroup name="tSensorActionAttributes">
    <xsd:attribute name="name" type="xsd:string" use="optional"/>
    <xsd:attribute name="enabled" type="xsd:boolean" use="optional"
default="true"/>
    <xsd:attribute name="filter" type="xsd:string"/>
    <xsd:attribute name="publishName" type="xsd:string" use="required"/>
    <xsd:attribute name="publishType" type="tns:tSensorActionPublishType"
use="required"/>
    <!--
      the name of the JMS Queue/Topic or custom java API, ignored for other
      publishTypes
    -->
    <xsd:attribute name="publishTarget" type="xsd:string" use="optional"/>
  </xsd:attributeGroup>

  <!--
    The sensor action type. A sensor action consists:
    + unique name
    + effective date
    + expiration date - Optional. If not defined, the probe is active
                        indefinitely.
    + filter (to potentially suppress data publishing even if a sensor marks
             it as interesting). - Optional. If not defined, no filter is
             used.
    + publishName A name of a publisher
    + publishType What to do with the sensor data?
    + publishTarget Name of a JMS Queue/Topic or custom publisher.
    + potentially many sensors.
  -->
  <xsd:complexType name="tSensorAction">
    <xsd:sequence>
      <xsd:element name="sensorName" type="xsd:string" minOccurs="1"
maxOccurs="unbounded"/>
      <xsd:element name="property" minOccurs="0" maxOccurs="unbounded"
type="tns:tSensorActionProperty"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="tns:tSensorActionAttributes"/>
  </xsd:complexType>
```

```
 <!--
   define a listing of sensor actions in a single document. It might be a good
idea  to
   have one sensor action list per business process.
 -->
 <xsd:complexType name="tSensorActionList">
   <xsd:sequence>
     <xsd:element name="action" type="tns:tSensorAction" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:simpleType name="tSensorKind">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="fault"/>
     <xsd:enumeration value="variable"/>
     <xsd:enumeration value="activity"/>
   </xsd:restriction>
 </xsd:simpleType>

 <xsd:complexType name="tActivityConfig">
   <xsd:annotation>
     <xsd:documentation>
       The configuration part of an activity sensor comprises of a mandatory
'evalTime' attribute
       and an optional list of variable configurations
     </xsd:documentation>
   </xsd:annotation>
   <xsd:complexContent>
     <xsd:extension base="tns:tSensorConfig">
       <xsd:sequence>
         <xsd:element name="variable" type="tns:tActivityVariableConfig"
maxOccurs="unbounded" minOccurs="0"/>
       </xsd:sequence>
       <xsd:attribute name="evalTime" type="xsd:string" use="required"/>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>

   <xsd:complexType name="tAdapterConfig">
     <xsd:annotation>
       <xsd:documentation>
         The configuration part of a adapter activity extends the activty
configuration with additional attributes for adapters
       </xsd:documentation>
     </xsd:annotation>
     <xsd:complexContent>
       <xsd:extension base="tns:tActivityConfig">
         <xsd:attribute name="headerVariable" use="required" type="xsd:string"/>
         <xsd:attribute name="partnerLink" use="required" type="xsd:string"/>
       <xsd:attribute name="portType" use="required" type="xsd:string"/>
       <xsd:attribute name="operation" use="required" type="xsd:string"/>
       </xsd:extension>
     </xsd:complexContent>
   </xsd:complexType>

 <xsd:complexType name="tVariableConfig">
   <xsd:complexContent>
     <xsd:extension base="tns:tSensorConfig">
       <xsd:attribute name="outputDataType" use="required" type="xsd:string"/>
```

```
      <xsd:attribute name="outputNamespace" use="required" type="xsd:string"/>
      <xsd:attribute name="queryName" use="optional" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="tActivityVariableConfig">
  <xsd:complexContent>
    <xsd:extension base="tns:tVariableConfig">
      <xsd:attribute name="target" type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="tFaultConfig">
  <xsd:complexContent>
    <xsd:extension base="tns:tSensorConfig"/>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="tNotificationSvcConfig">
  <xsd:complexContent>
    <xsd:extension base="tns:tActivityConfig">
      <xsd:attribute name="inputVariable" use="required" type="xsd:string"/>
      <xsd:attribute name="outputVariable" use="required" type="xsd:string"/>
      <xsd:attribute name="operation" use="required" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="tSensorConfig">
  <xsd:sequence>
    <xsd:element name="action" type="tns:tInlineSensorAction" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="tInlineSensorAction">
  <xsd:complexContent>
    <xsd:restriction base="tns:tSensorAction"/>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:complexType name="tSensor">
  <xsd:sequence>
    <xsd:element name="activityConfig" type="tns:tActivityConfig"
minOccurs="0"/>
    <xsd:element name="adapterConfig" type="tns:tAdapterConfig" minOccurs="0"/>
    <xsd:element name="faultConfig" type="tns:tFaultConfig" minOccurs="0"/>
    <xsd:element name="notificationConfig" type="tns:tNotificationSvcConfig"
minOccurs="0"/>
    <xsd:element name="variableConfig" type="tns:tVariableConfig"
minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="sensorName" use="required" type="xsd:string"/>
  <xsd:attribute name="kind" use="required" type="tns:tSensorKind"/>
  <xsd:attribute name="classname" use="required" type="xsd:string"/>
  <xsd:attribute name="target" use="required" type="xsd:string"/>
 </xsd:complexType>
```

```
 <xsd:complexType name="tSensorList">
   <xsd:sequence>
     <xsd:element name="sensor" type="tns:tSensor" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="tRouterData">
   <xsd:sequence>
     <xsd:element name="header" type="tns:tHeaderInfo"/>
     <xsd:element name="payload" type="tns:tSensorData"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="tHeaderInfo">
   <xsd:sequence>
     <xsd:element name="processName" type="xsd:string"/>
     <xsd:element name="processRevision" type="xsd:string"/>
     <xsd:element name="domain" type="xsd:string"/>
     <xsd:element name="instanceId" type="xsd:integer"/>
     <xsd:element name="midTierInstance" type="xsd:string"/>
     <xsd:element name="timestamp" type="xsd:dateTime"/>
     <xsd:element name="sensor" type="tns:tSensor"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="tSensorData">
     <xsd:sequence>
       <xsd:element name="activityData" type="tns:tActivityData" minOccurs="0"/>
     <xsd:element name="faultData" type="tns:tFaultData" minOccurs="0"/>
     <xsd:element name="adapterData" minOccurs="0" type="tns:tAdapterData"/>
       <xsd:element name="variableData" type="tns:tVariableData" minOccurs="0"
maxOccurs="unbounded"/>
       <xsd:element name="notificationData" type="tns:tNotificationData"
minOccurs="0"/>
     </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="tFaultData">
   <xsd:sequence>
     <xsd:element name="activityName" type="xsd:string"/>
     <xsd:element name="activityType" type="xsd:string"/>
     <xsd:element name="data" type="xsd:anyType" minOccurs="0"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="tActivityData">
   <xsd:sequence>
     <xsd:element name="activityType" type="xsd:string"/>
     <xsd:element name="evalPoint" type="xsd:string"/>
     <xsd:element name="errorMessage" nillable="true" minOccurs="0"
type="xsd:string"/>
   </xsd:sequence>
 </xsd:complexType>

 <!--
  xml type that is provided to sensors for variable Datas. Note the
     any element represents variable data.
  -->
 <xsd:complexType name="tVariableData">
```

```
    <xsd:sequence>
      <xsd:element name="target" type="xsd:string"/>
      <xsd:element name="queryName" type="xsd:string"/>
      <xsd:element name="updaterName" type="xsd:string" minOccurs="1"/>
      <xsd:element name="updaterType" type="xsd:string" minOccurs="1"/>
      <xsd:element name="data" type="xsd:anyType"/>
      <xsd:element name="dataType" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="tNotificationData">
    <xsd:complexContent>
      <xsd:extension base="tns:tActivityData">
        <xsd:sequence>
          <xsd:element name="messageID" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
          <xsd:element name="fromAddress" type="xsd:string" minOccurs="0"/>
          <xsd:element name="toAddress" type="xsd:string" minOccurs="0"/>
          <xsd:element name="deliveryChannel" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>

  </xsd:complexType>
  <xsd:complexType name="tAdapterData">
    <xsd:complexContent>
      <xsd:extension base="tns:tActivityData">
        <xsd:sequence>
          <xsd:element name="endpoint" type="xsd:string"/>
          <xsd:element name="direction" type="xsd:string"/>
          <xsd:element name="adapterType" type="xsd:string"/>
          <xsd:element name="priority" type="xsd:string" minOccurs="0"/>
          <xsd:element name="messageSize" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <!--
    The header of the document contains some metadata.
  -->
  <xsd:complexType name="tSensorActionHeader">
    <xsd:sequence>
      <xsd:element name="processName" type="xsd:string"/>
      <xsd:element name="processVersion" type="xsd:string"/>
      <xsd:element name="processID" type="xsd:long"/>
      <xsd:element name="midTierInstance" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="actionName" use="required" type="xsd:string"/>
  </xsd:complexType>

  <!--
Sensor Action data is presented in the form of a header and potentially many
data elements depending on how many sensors associated to the sensor action
marked the data as interesting.
  -->
  <xsd:complexType name="tSensorActionData">
    <xsd:sequence>
      <xsd:element name="header" type="tns:tHeaderInfo"/>
      <xsd:element name="payload" type="tns:tSensorData" minOccurs="1"
                   maxOccurs="1"/>
```

```
    </xsd:sequence>
  </xsd:complexType>
<!--
  <xsd:simpleType name="tActivityEvalPoint">
    <xsd:restriction>
      <xsd:enumeration value="start"/>
      <xsd:enumeration value="complete"/>
      <xsd:enumeration value="fault"/>
      <xsd:enumeration value="compensate"/>
      <xsd:enumeration value="retry"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="tNotificationAction">
    <xsd:restriction>
      <xsd:enumeration value="creation"/>
      <xsd:enumeration value="statusUpdate"/>
    </xsd:restriction>
  </xsd:simpleType>
-->

  <!--
    The process sensor value header comprises of a timestamp
    where the sensor was triggered and the sensor metadata
  -->
  <xsd:complexType name="tProcessSensorValueHeader">
    <xsd:sequence>
      <xsd:element name="timestamp" type="xsd:dateTime"/>
      <xsd:element ref="tns:sensor"/>
    </xsd:sequence>
  </xsd:complexType>

  <!--
    Extend tActivityData to include more elements
  -->
  <xsd:complexType name="tProcessActivityData">
    <xsd:complexContent>
      <xsd:extension base="tns:tActivityData">
        <xsd:sequence>
          <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
 maxOccurs="1"/>
          <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
 maxOccurs="1"/>
          <xsd:element name="evalTime" type="xsd:long" minOccurs="0"
 maxOccurs="1"/>
          <xsd:element name="retryCount" type="xsd:int" minOccurs="0"
 maxOccurs="1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!--
    Extend tVariableData to include more elements
  -->
  <xsd:complexType name="tProcessVariableData">
    <xsd:complexContent>
      <xsd:extension base="tns:tVariableData">
        <xsd:sequence>
          <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
```

```
maxOccurs="1"/>
         <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>

 <!--
   Extend tFaultData to include more elements
 -->
 <xsd:complexType name="tProcessFaultData">
   <xsd:complexContent>
     <xsd:extension base="tns:tFaultData">
       <xsd:sequence>
         <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
         <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>

 <!--
   Extend tAdapterData to include more elements
 -->
 <xsd:complexType name="tProcessAdapterData">
   <xsd:complexContent>
     <xsd:extension base="tns:tAdapterData">
       <xsd:sequence>
         <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
         <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>

 <!--
   Extend tNotificationData to include more elements
 -->
 <xsd:complexType name="tProcessNotificationData">
   <xsd:complexContent>
     <xsd:extension base="tns:tNotificationData">
       <xsd:sequence>
         <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
         <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>
 <!--
   Copy of tSensorData type with some modified types.
 -->
 <xsd:complexType name="tProcessSensorData">
   <xsd:sequence>
```

```
     <xsd:element name="activityData" type="tns:tProcessActivityData"
minOccurs="0"/>
     <xsd:element name="faultData" type="tns:tProcessFaultData" minOccurs="0"/>
     <xsd:element name="adapterData" minOccurs="0"
type="tns:tProcessAdapterData"/>
     <xsd:element name="variableData" type="tns:tProcessVariableData"
minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element name="notificationData" type="tns:tProcessNotificationData"
minOccurs="0"/>
   </xsd:sequence>
 </xsd:complexType>
 <!--
   A single process sensor value comprises of the sensor value metadata
   (sensor and timestamp) and the payload (the value) of the sensor
 -->
 <xsd:complexType name="tProcessSensorValue">
   <xsd:sequence>
     <xsd:element name="header" type="tns:tProcessSensorValueHeader"/>
     <xsd:element name="payload" type="tns:tProcessSensorData"/>
   </xsd:sequence>
 </xsd:complexType>

 <!--
   Process instance header.
 -->
 <xsd:complexType name="tProcessInstanceInfo">
   <xsd:sequence>
     <xsd:element name="processName" type="xsd:string"/>
     <xsd:element name="processRevision" type="xsd:string"/>
     <xsd:element name="domain" type="xsd:string"/>
     <xsd:element name="instanceId" type="xsd:integer"/>
   </xsd:sequence>
 </xsd:complexType>

 <!--
   The list of sensor values comprises of a process header describing the
   BPEL process with name, cube instance id etc. and a list of sensor values
   comprising of sensor metadata information and sensor values.
 -->
 <xsd:complexType name="tProcessSensorValueList">
   <xsd:sequence>
     <xsd:element name="process" type="tns:tProcessInstanceInfo" minOccurs="1"
maxOccurs="1"/>
     <xsd:element name="sensorValue" type="tns:tProcessSensorValue" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
 </xsd:complexType>

 <!-- The sensor list is the root element of the sensor.xml document in the
      bpel process suitcase and is used to define sensors. -->
 <xsd:element name="sensors" type="tns:tSensorList"/>

 <!-- A sensor is used to monitor a particular aspect of a bpel process -->
 <xsd:element name="sensor" type="tns:tSensor"/>

 <!-- The actions element is the root element of the sensorAction.xml document
      in the bpel process suitcase and is used to define sensor actions.
      Sensor actions define how to publish data captured by sensors -->
 <xsd:element name="actions" type="tns:tSensorActionList"/>
```

```
  <!-- actionData elements are produced by the sensor framework and sent to the
       appropriate data publishers when sensors 'fire' -->
  <xsd:element name="actionData" type="tns:tSensorActionData"/>

  <!-- This element is used when the client API is used to query sensor values
       stored in the default reports schema -->
  <xsd:element name="sensorValues" type="tns:tProcessSensorValueList"/>
</xsd:schema>
```

E

Oracle BAM Web Services Operations

This appendix is a reference for the operations provided by the Oracle BAM DataObjectOperations and DataObjectDefinition web services. More information about the Oracle BAM web services is available in Chapter 56, "Using Oracle BAM Web Services."

This appendix includes the following sections:

- Section E.1, "DataObjectOperations10131"
- Section E.2, "DataObjectOperationsByName"
- Section E.3, "DataObjectOperationsByID"
- Section E.4, "DataObjectDefinition Operations"
- Section E.5, "ManualRuleFire Operations"

E.1 DataObjectOperations10131

The following operations are supported by the DataObjectOperations10131 web service:

- Section E.1.1, "Batch"
- Section E.1.2, "Delete"
- Section E.1.3, "Insert"
- Section E.1.4, "Update"
- Section E.1.5, "Upsert"

E.1.1 Batch

Batch performs batch operations on a data object.

E.1.1.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

xmlPayload (xsd:string)

Contains the batch payload for any operations to be performed. For example:

```
<payload>
  <_Employees operation="insert">
    <_Salesperson>Tim Bray</_Salesperson>
    <_Sales_Area>EMEA</_Sales_Area>
    <_Sales_Number>12345</_Sales_Number>
  </_Employees>
  <_Employees operation="update" keys="_Sales_Number">
    <_Salesperson>Tim Bray</_Salesperson>
    <_Sales_Area>EMEA</_Sales_Area>
    <_Sales_Number>12345</_Sales_Number>
  </_Employees>
</payload>
```

E.1.2 Delete

Delete removes a row from the data object.

E.1.2.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number,_Sales_Area
```

xmlPayload (xsd:string)

Payload for the `where` clause to delete rows in a data object. For example:

```
<_Employees>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.1.3 Insert

Insert adds rows to the specified data object.

E.1.3.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

xmlPayload (xsd:string)

The payload is specific to each data object.

```
<_Employees>
  <_Salesperson>Time Bray</_Salesperson>
```

```
  <_Sales_Area>EMEA</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.1.4 Update

Update operation updates existing data with new data in a data object.

E.1.4.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number,_Sales_Area
```

xmlPayload (xsd:string)

Payload for the update statement and where clause to update rows in a data object. For example:

```
<_Employees>
  <_Sales_Area>Asia Pacific</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.1.5 Upsert

Upsert operation updates existing data with new data in an existing row in a data object. If the row does not exist a new row is created.

E.1.5.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number,_Sales_Area
```

xmlPayload (xsd:string)

Payload for the insert or update statement and where clause to upsert rows in a data object. For example:

```
<_Employees>
  <_Salesperson>Time Bray</_Salesperson>
  <_Sales_Area>EMEA</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.2 DataObjectOperationsByName

The following operations are supported by the DataObjectOperations10131, DataObjectOperationsByName, and DataObjectOperationsByID web services.

- Section E.2.1, "Delete"
- Section E.2.2, "Get"
- Section E.2.3, "Insert"
- Section E.2.4, "Update"
- Section E.2.5, "Upsert"

E.2.1 Delete

Delete removes a row from the data object.

E.2.1.1 Request Message

The request message contains the following parameters.

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
Sales Number, Sales Area
```

xmlPayload (xsd:string)

Payload for the where clause to delete rows in a data object. For example:

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
    <Row>
      <Column Name="Salesperson" Value="Greg Guan" />
    </Row>
  </Contents>
</DataObject>
```

E.2.2 Get

Get fetches the details from a data object per the specifications in the XML payload

Get is only available in DataObjectOperationsByName web service.

E.2.2.1 Request Message

The request message contains the following parameters.

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
Sales Number, Sales Area
```

xmlPayload (xsd:string)

The payload specifies what to get from the data object.

For the DataObjectOperationsByName web service the data object name is specified in the payload, for example:

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
```

```
    <Row>
      <Column Name="Salesperson" Value="Greg Masters"/>
    </Row>
  </Contents>
</DataObject>
```

E.2.3 Insert

Insert adds rows to the specified data object.

E.2.3.1 Request Message

The request message contains the following parameters.

xmlPayload (xsd:string)

The payload is specific to each data object.

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
    <Row>
      <Column Name="Salesperson" Value="Greg Guan" />
      <Column Name="Sales Area" Value="Northeast" />
      <column Name="Sales Number" Value="5671" />
    </Row>
  </Contents>
</DataObject>
```

E.2.4 Update

Update operation updates existing data with new data in a data object.

E.2.4.1 Request Message

The request message contains the following parameters.

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
Sales Number, Sales Area
```

xmlPayload (xsd:string)

Payload for the update statement and where clause to update rows in a data object. For example:

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
    <Row>
      <Column Name="Salesperson" Value="Greg Guan" />
    </Row>
  </Contents>
</DataObject>
```

E.2.5 Upsert

Upsert operation updates existing data with new data in an existing row in a data object. If the row does not exist a new row is created.

E.2.5.1 Request Message

The request message contains the following parameters.

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
Sales Number, Sales Area
```

xmlPayload (xsd:string)

Payload for the insert or update statement and where clause to upsert rows in a data object. For example:

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
    <Row>
      <Column Name="Salesperson" Value="Greg Guan" />
      <Column Name="Sales Area" Value="Northeast" />
      <column Name="Sales Number" Value="5671" />
    </Row>
  </Contents>
</DataObject>
```

E.3 DataObjectOperationsByID

The following operations are supported by the DataObjectOperations10131, DataObjectOperationsByName, and DataObjectOperationsByID web services.

- Section E.3.1, "Batch"
- Section E.3.2, "Delete"
- Section E.3.3, "Insert"
- Section E.3.4, "Update"
- Section E.3.5, "Upsert"

E.3.1 Batch

Batch performs batch operations on a data object.

E.3.1.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

xmlPayload (xsd:string)

Contains the batch payload for any operations to be performed. For example:

```
<payload>
  <_Employees operation="insert">
    <_Salesperson>Tim Bray</_Salesperson>
    <_Sales_Area>EMEA</_Sales_Area>
    <_Sales_Number>12345</_Sales_Number>
  </_Employees>
  <_Employees operation="update" keys="_Sales_Number">
```

```
    <_Salesperson>Tim Bray</_Salesperson>
    <_Sales_Area>EMEA</_Sales_Area>
    <_Sales_Number>12345</_Sales_Number>
  </_Employees>
</payload>
```

E.3.2 Delete

Delete removes a row from the data object.

E.3.2.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

This parameter is not required by the DataObjectOperationsByName web service because the data object name and path are part of the payload.

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number,_Sales_Area
```

xmlPayload (xsd:string)

Payload for the where clause to delete rows in a data object. For example:

```
<_Employees>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.3.3 Insert

Insert adds rows to the specified data object.

E.3.3.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

xmlPayload (xsd:string)

The payload is specific to each data object.

For the DataObjectOperationsByName web service the data object name is specified in the payload, for example:

```
<_Employees>
  <_Salesperson>Time Bray</_Salesperson>
  <_Sales_Area>EMEA</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.3.4 Update

Update operation updates existing data with new data in a data object.

E.3.4.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number,_Sales_Area
```

xmlPayload (xsd:string)

Payload for the update statement and where clause to update rows in a data object. For example:

```
<_Employees>
  <_Sales_Area>Asia Pacific</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.3.5 Upsert

Upsert operation updates existing data with new data in an existing row in a data object. If the row does not exist a new row is created.

E.3.5.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number,_Sales_Area
```

xmlPayload (xsd:string)

Payload for the insert or update statement and where clause to upsert rows in a data object. For example:

```
<_Employees>
  <_Salesperson>Time Bray</_Salesperson>
  <_Sales_Area>EMEA</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.4 DataObjectDefinition Operations

The following operations are supported by DataObjectDefinition web service.

- Section E.4.1, "Create"
- Section E.4.2, "Delete"
- Section E.4.3, "Get"
- Section E.4.4, "Update"

E.4.1 Create

Create creates a new data object. By specifying columnar elements, you can create calculated and lookup fields in addition to regular fields ass show in the examples.

E.4.1.1 Request Message

The request message contains the following parameter.

xmlPayload (xsd:string)

Contains the payload to create a data object.

Table E–1 xmlPayload Elements and Descriptions and Valid Values

Element	Description and Values
/DataObject/@External	0 (zero) indicates that the data object is not from an external data source (default). 1 indicates that the data object is from an external data source.
/DataObject/@Name	Name of the data object to be created not including the directory path.
/DataObject/@Path	Directory path in which to create the data object.
/DataObject/@Version	Data objects can be versioned 0 (default) through 14.
/DataObject/@TipText	Description of the data object to be created.
/DataObject/Layout/Column/@Name	Name of the column (field) in the data object.
/DataObject/Layout/Column/@Type	The following values are valid for column type: auto-incr-integer boolean calculated clob datetime decimal float iterID integer lookup string timestamp
/DataObject/Layout/Column/@Nullable	1 (default) indicates that the column supports null values. 0 (zero) indicates that the column does not support null values.
/DataObject/Layout/Column/@Public	1 (default) indicates that the column is public. 0 (zero) indicates that the column is not public.

Table E–1 (Cont.) xmlPayload Elements and Descriptions and Valid Values

Element	Description and Values
/DataObject/Layout/Column/@MaxSize	For string type columns, this attribute specifies the maximum permissible string size. Default value is 30.
/DataObject/Layout/Column/@Precision	For decimal type columns, this attribute specifies the precision of the decimal value.
/DataObject/Layout/Column/@Scale	For decimal type columns, this attribute specifies the scale of the decimal value.
/DataObject/Layout/Column/@TipText	Column description

Example E–1 xmlPayload to Create Data Object With Regular Columns

```
<DataObject Version="14" Name="Employees3" ID="_Employees3" Path="/Samples"
            External="0">
  <Layout>
    <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="30"
            Nullable="1" Public="1" />
    <Column Name="Sales Number" ID="_Sales_Number" Type="decimal"
            Nullable="1" Public="1" />
    <Column Name="Timestamp" ID="_Timestamp" Type="timestamp"
            Nullable="0" Public="1" />
    <Indexes />
  </Layout>
</DataObject>
```

Example E–2 xmlPayload to Create Data Object With Lookup Field

```
<DataObject Version="14" Name="LookupDO" ID="_LookupDO" Path="/Samples">
  <Layout>
    <Description><![CDATA[Lookup]]></Description>
    <Column Name="Name" ID="_Name" Type="string" MaxSize="100"
            Nullable="1" Public="1" />
    <Column Name="ID" ID="_ID" Type="integer" Nullable="1" Public="1" />
    <Column Name="Sales Area" ID="_Sales_Area" Type="lookup">
      <Lookup>
        <DataObject>
          <ID>_Employees</ID>
          <Path>/Samples</Path>
        </DataObject>
        <LookupFieldID>_Sales_Area</LookupFieldID>
        <MatchFields>
          <KeyPair>
            <PrimaryKeyID>_Sales_Number</PrimaryKeyID>
            <ForeignKeyID>_ID</ForeignKeyID>
          </KeyPair>
        </MatchFields>
      </Lookup>
    </Column>
    <Indexes />
  </Layout>
</DataObject>
```

Note that when you construct the XML payload for the Create operation, and the data object version is lower than 12, use PrimaryKey instead of PrimaryKeyID, ForeignKey instead of ForeignKeyID, LookupField instead of LookupFieldID, and provide name values instead of IDs for those fields.

Example E–3 xmlPayload to Create Data Object With Calculated Field

```
<DataObject Version="14" Name="CalculatedDO" ID="_CalculatedDO" Path="/Samples">
  <Layout>
    <Description><![CDATA[Calculated Column]]></Description>
    <Column Name="Name" ID="_Name" Type="string" MaxSize="100" Nullable="1"
            Public="1" />
    <Column Name="Address" ID="_Address" Type="string" MaxSize="100" Nullable="1"
            Public="1" />
    <Column Name="Salary" ID="_Salary" Type="decimal" Scale="10" Nullable="1"
            Public="1" />
    <Column Name="Income Tax" ID="_Income_Tax" Type="calculated"
            CalculatedExpression="&lt;expression type=&quot;MathExpression&quot;
&gt;&lt;operation&gt;&lt;left&gt;&lt;type&gt;FieldID&lt;/type&gt;&lt;ivalue&gt;
_Salary&lt;/ivalue&gt;&lt;/left&gt;&lt;operator&gt;*&lt;/operator&gt;&lt;right&gt;
&lt;type&gt;DECIMAL&lt;/type&gt;&lt;ivalue&gt;0.3&lt;/ivalue&gt;&lt;/right&gt;&lt;
/operation&gt;&lt;/expression&gt;" ExpressionUserText="(Salary * 0.3)" />
    <Indexes />
  </Layout>
</DataObject>
```

E.4.1.2 Response Message

void

E.4.2 Delete

Delete removes a data object definition and its contents.

E.4.2.1 Request Message

The request message contains the following parameter.

dataObjectFullName (xsd:string)

Full relative path and name of the data object to be deleted. For example:

```
/Samples/Employees
```

E.4.2.2 Response Message

void

E.4.3 Get

Get retrieves an existing data object definition.

E.4.3.1 Request Message

The request message contains the following parameters.

dataObjectFullName (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Sales
```

E.4.3.2 Response Message

The response message contains the following parameter.

xmlPayload (xsd:string)

An XML description of the data object is returned. The schema used is the same definition as described for the Create and Update operations. You can use this operation to find the ID values of the data object and any columns.

Example E–4 xmlPayload for Get Operation

```
<DataObject Version="14" Name="Employees" Path="/Samples" External="0">
  <Layout>
    <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
            Nullable="1" Public="1" />
    <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
            Nullable="1" Public="1" />
    <Column Name="Sales Number" ID="_Sales_Number" Type="integer" Nullable="1"
            Public="1" />
    <Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0" />
            Public="1" />
    <Indexes />
  </Layout>
</DataObject>
```

E.4.4 Update

Update updates the definition of an existing data object. If a specified column exists in the original definition, the new column definition overwrites the old one. If columns in the existing definition are not specified in the new definition, their definitions are removed. The data object index definition can be updated as well.

E.4.4.1 Request Message

The request message contains the following parameters.

xmlPayload (xsd:string)

Payload for the Update operation is similar to the Create payload with one additional attribute. For example:

```
<DataObject Version="14" Name="Employees4" ID="_Employees4" Path="/Samples"
External="0">
  <Layout>
    <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="50"
            Nullable="1" Public="1" />
    <Column Name="Sales Number" ID="_Sales_Number" Type="integer"
            Nullable="1" Public="1" />
    <Column Name="Timestamp" ID="_Timestamp" Type="timestamp"
            Nullable="0" Public="1" />
    <Indexes />
  </Layout>
</DataObject>
```

E.4.4.2 Response Message

void

E.5 ManualRuleFire Operations

The following operation is supported by ManualRuleFire web service.

- Section E.5.1, "FireRuleByName"

E.5.1 FireRuleByName

Use this operation to manually launch a rule.

This web service takes a string parameter, which should have user name, followed by a period (.), followed by the alert name. For example:

```
user_name.alertname
```

The period is used as a separator between the user name and the alert name. The web service always treats last period in the string as the separator, allowing the user name to contain periods. For example

```
user.nema.alerrtname
```

It follows then that the alert names cannot contain a period. If you must use the ManualRuleFire web service with an alert containing a period in its name, the alert must be renamed so that it does not contain any periods.

E.5.1.1 Request Message

The request message contains the following parameter.

xmlPayload (xsd:string)

An example:

```
<FireRuleByName xmlns="http://xmlns.oracle.com/bam">
  <strRuleName>string</strRuleName>
</FireRuleByName>
```

E.5.1.2 Response Message

Returns (xsd:string)

```
<FireRuleByNameResponse xmlns="http://xmlns.oracle.com/bam">
  <FireRuleByNameResult>string</FireRuleByNameResult>
</FireRuleByNameResponse>
```

F

Oracle BAM Alert Rule Options

This appendix describes the options for creating alert rules.

This appendix includes the following sections:

F.1 Events

Events launch the rule and trigger the action. Each rule contains only one event. Oracle BAM provides the following events:

- In a specific amount of time
- At a specific time today
- On a certain day at a specific time
- Every interval between two times
- Every date interval starting on certain date at a specific time
- When a report changes
- When a data field changes in data object
- When a data field in a report meets specified conditions
- When a data field in a data object meets specified conditions
- When this rule is launched

F.1.1 In a specific amount of time

When you select the event **In a specific amount of time**, you must complete the rule expression by selecting a time interval in seconds, minutes, or hours.

F.1.2 At a specific time today

When you select the event **At a specific time today**, you must complete the rule expression by selecting the time at which to launch the alert.

F.1.3 On a certain day at a specific time

When you select the event **On a certain day at a specific time**, you must complete the rule expression by selecting both the date and the time at which to launch the alert.

F.1.4 Every interval between two times

When you select the event **Every interval between two times**, you must complete the rule expression by configuring the following settings.

- **select time interval**

  Set the number of minutes, hours, or days between each alert launch.

- **select time**

  Set the times of day between which the rule is valid and the alert is launched.

F.1.5 Every date interval starting on certain date at a specific time

When you select the event **Every date interval starting on a certain date at a specific time**, you must complete the rule expression by configuring the following settings.

- **select date interval**

  Set the alert to launch every Day, Week, Month, or Year.

- **select date**

  Set the date on which the rule is valid and the alert is launched.

- **select time**

  Set the time of day at which the rule is valid and the alert is launched.

F.1.6 When a report changes

When a report changes is launched when runtime changes in a report occur (not changes in the report definition), that is every time a change list is delivered to the report from the Oracle BAM Server. Report changes can include changes to data in data objects and changes due to Active Now settings.

When you select the event **When a report changes**, you must complete the rule expression by configuring the following settings.

- **select report**

  Select the report to monitor for changes.

- run as **<user_name>** (This option appears only if the user creating the alert is a member of the administrator role.)

  Select the Oracle BAM user who the selected report runs as. You can select only one run as user. The default run as user is the logged in Oracle BAM user who is creating the alert.

  Only recipients who have security permissions that are the same or higher than the run as user receive the notification for report changes, honoring row level security as implemented by the Oracle BAM Architect in the data objects used in the report.

  Names that are preceded with a hash (#) are distribution lists.

  If there are changes in a report's data object rows that none of the alert recipients have permissions to access, no recipients are notified.

F.1.7 When a data field changes in data object

When you select the event **When a data field changes in a data object**, you must complete the rule expression by configuring the following settings.

> **Note:** The event **When a data field in a data object meets specified conditions** responds only to row inserts and row updates, but it does not respond to row deletes; however, the event **When a data field changes in a data object** responds to row deletes.

- **select data field**

  Select the data object field to monitor for changes. In the Field Selection dialog box, locate the data object in the top left section of the dialog box, then select the field in the top right section of the dialog box. Finally, select one or more fields to group by and an aggregate function for the selected field.

- run as **<user_name>** (This option appears only if the user creating the alert is a member of the administrator role.)

  Select the Oracle BAM user who the selected report runs as. You can select only one run as user. The default run as user is the logged in Oracle BAM user who is creating the alert.

  Only recipients who have security permissions that are the same or higher than the run as user receive the notification for report changes, honoring row level security as implemented by the Oracle BAM Architect in the data objects used in the report.

  Names that are preceded with a hash (#) are distribution lists.

  If there are changes in a report's data object rows that none of the alert recipients have permissions to access, no recipients are notified.

F.1.8 When a data field in a report meets specified conditions

When you select the event **When a data field changes in a data object**, you must complete the rule expression by configuring the following settings.

- **select report**

  Select the report to monitor for changes.

- **this data field has a condition of x**

  In the Alert Rule Editor dialog box, select the data object to monitor. Then you can set the condition under which the alert should fire.

  - **Row Filter** - Create a filter on a field in the data object to express a condition that, when met, launches the rule. All of the functionality available in report filters is provided. See "Filtering Data" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for more information.
  - **Group Filter** - The Group Filter is similar to the Row Filter in that it provides all of the filtering functionality available in report filters. The special feature here is that it allows filters to be created on a field where a summary function has been applied. See "Filtering Data" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for more information about building filter expressions.

- **Group** - Choose one or more fields on which to create a grouping, adding further complexity to any filters created in the Row Filter or Group Filter tabs.

- run as **<user_name>** (This option appears only if the user creating the alert is a member of the administrator role.)

  Select the Oracle BAM user who the selected report runs as. You can select only one run as user. The default run as user is the logged in Oracle BAM user who is creating the alert.

  Only recipients who have security permissions that are the same or higher than the run as user receive the notification for report changes, honoring row level security as implemented by the Oracle BAM Architect in the data objects used in the report.

  Names that are preceded with a hash (#) are distribution lists.

  If there are changes in a report's data object rows that none of the alert recipients have permissions to access, no recipients are notified.

F.1.9 When a data field in a data object meets specified conditions

When you select the event **When a data field in a data object meets specified condition**, you must complete the rule expression by configuring the following settings.

> **Note:** The event **When a data field in a data object meets specified conditions** responds only to row inserts and row updates, but it does not respond to row deletes; however, the event **When a data field changes in a data object** responds to row deletes.

- **this data field has a condition of x**

  In the Alert Rule Editor dialog box, select the data object to monitor. Then you can set the condition under which the alert should fire.

  - **Row Filter** - Create a filter on a field in the data object to express a condition that, when met, launches the rule. All of the functionality available in report filters is provided. See "Filtering Data" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for more information.

  - **Group Filter** - The Group Filter is similar to the Row Filter in that it provides all of the filtering functionality available in report filters. The special feature here is that it allows filters to be created on a field where a summary function has been applied. See "Filtering Data" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for more information about building filter expressions.

  - **Group** - Choose one or more fields on which to create a grouping, adding further complexity to any filters created in the Row Filter or Group Filter tabs.

- run as **<user_name>** (This option appears only if the user creating the alert is a member of the administrator role.)

  Select the Oracle BAM user who the selected report runs as. You can select only one run as user. The default run as user is the logged in Oracle BAM user who is creating the alert.

  Names that are preceded with a hash (#) are distribution lists.

Only recipients who have security permissions that are the same or higher than the run as user receive the notification for report changes, honoring row level security as implemented by the Oracle BAM Architect in the data objects used in the report.

If there are changes in a report's data object rows that none of the alert recipients have permissions to access, no recipients are notified.

F.1.10 When this rule is launched

The event **When this rule is launched** is used to create a rule dependent on another rule which uses the **Launch a rule** action. Several rules can be created using **When this rule is launched** in a hierarchy.

F.2 Conditions

Conditions are optional settings for constraining the time period in which the alert is fired. You can select any number and combination of conditions. Oracle BAM provides the following conditions:

- If it is between two times
- If It is between two days
- If it is a particular day of the week

F.2.1 If it is between two times

Select two times between which the rule should launch.

F.2.2 If It is between two days

Select two dates between which the rule should launch.

F.2.3 If it is a particular day of the week

Select a day of the week on which the rule should launch.

F.3 Actions

Actions are the results of the conditions and events of the rule expression having been met. You can configure any number and combination of actions. Oracle BAM provides the following actions:

- Send a report via email
- Send a message via email
- Send a report via email and escalate to another user after a specific amount of time
- Send a parameterized message
- Send a parameterized message for every matching row in a data object
- Launch a rule
- Launch rule if an action fails
- Delete rows from a Data Object
- Call a Web Service

- Run an Oracle Data Integrator Scenario
- Call an External Action

F.3.1 Send a report via email

Select a report, select to send the report as a report link or as a rendered report, and select a recipient.

Recipients can be selected from Oracle BAM users, or, if a property is set in Oracle BAM, external e-mail accounts. See Section 57.9, "Sending Alerts to External E-mail Accounts" for more information.

F.3.2 Send a message via email

Create an email message to send and select a recipient.

Recipients can be selected from Oracle BAM users, or, if a property is set in Oracle BAM, external e-mail accounts. See Section 57.9, "Sending Alerts to External E-mail Accounts" for more information.

F.3.3 Send a report via email and escalate to another user after a specific amount of time

Select a report to send to the specified user. Select a secondary recipient to receive the message if the first recipient does not respond within the specified time period. The secondary recipient can be a single user or a distribution list.

When the condition of the alert rule is met, a report link is sent to the recipient. To respond to this alert, the recipient must click the report link and view the report. If the recipient does not view the report, it is escalated to the secondary user (or distribution list).

Recipients can be selected from Oracle BAM users, or, if a property is set in Oracle BAM, external e-mail accounts. See Section 57.9, "Sending Alerts to External E-mail Accounts" for more information.

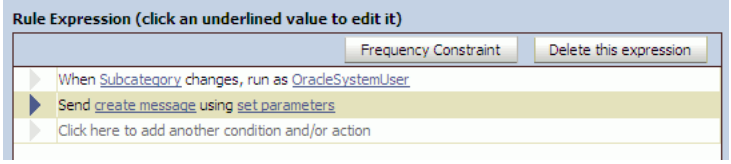
F.3.4 Send a parameterized message

This option enables you to email reports that require parameter inputs to Oracle BAM users. This action enables you to create a fully configurable email message and the parameter values that are passed to the report.

For information about creating prompts and parameters in Oracle BAM dashboards see "Using Prompts and Parameters" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring*.

You can use this option to send reports to other users under the conditions specified in the alert message. This action is available for the events **When a data field changes in data object** and **When a data field in a data object meets specified conditions**.

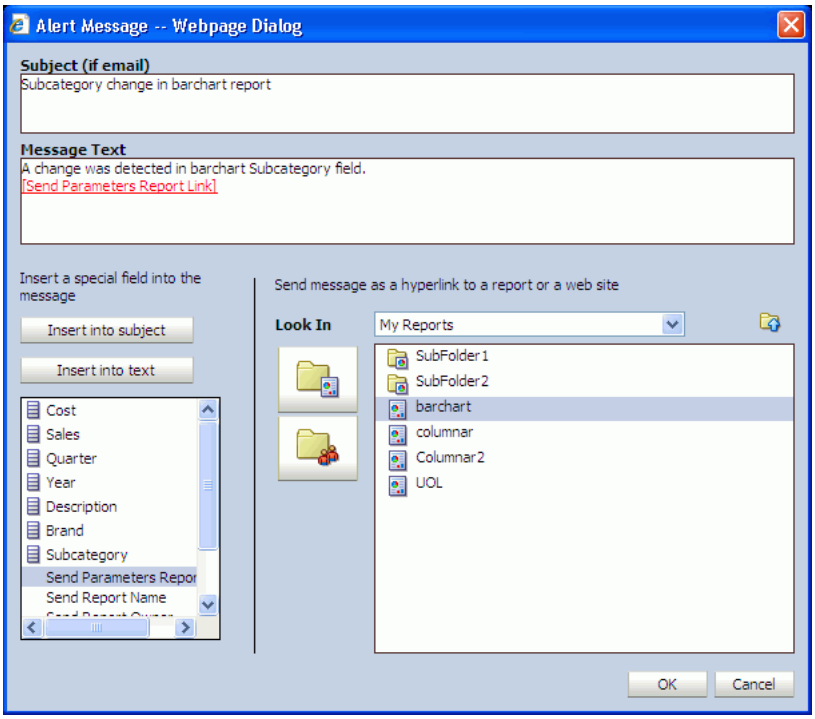
There are two properties that must be configured in this alert action: create message and set parameters.



To create the message

1. Click **create message** in the rule expression.
2. Enter a subject and message to send to the recipient. You can also select links to reports to send in the message body as shown in Figure F–1.

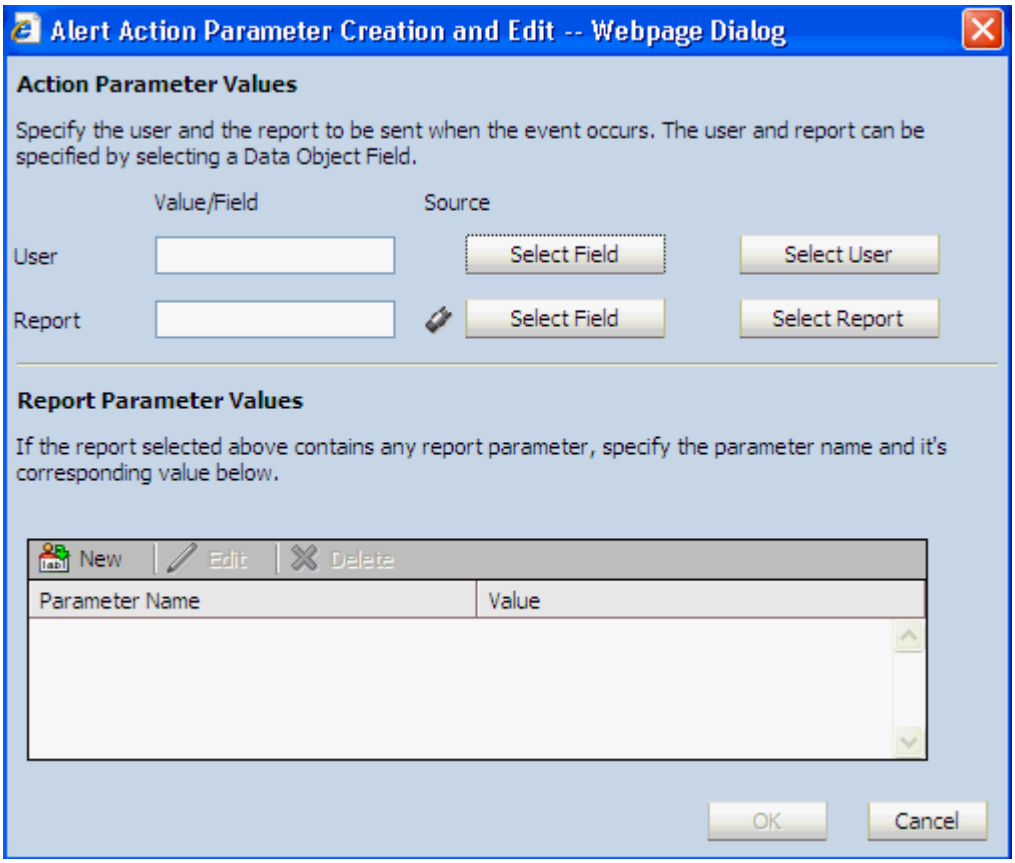
Figure F–1 Alert Message dialog box



To configure the parameter values that are passed to the report when it is opened by the recipient:

1. Click **set parameters** in the rule expression.
2. In the Alert Action Parameter Creation and Edit dialog box, populate the **User**, **Delivery**, and **Report** fields with either predefined values or dynamically from a Data Object field. Use the buttons to set the field values. **Select Field** enables you to select a field in a data object as a value.

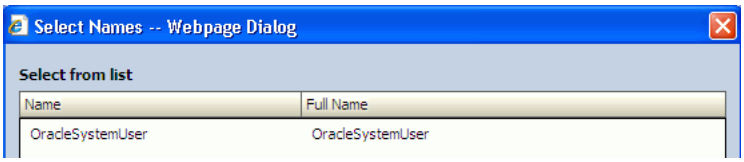
Figure F–2 Alert Action Parameter Creation and Edit dialog box



- **User** field

  If you populate this field using the Select User button, the recipients are selected from Oracle BAM users listed in Oracle BAM Administrator as shown in Figure F–3.

Figure F–3 Select Names dialog box



- **Report** field

  If you populate this field with the **Select Report** button, the value that appears in this field is the display name of the report.

  If you populate this field from a Data Object, the value must be the report ID of that report, and not the display name. To get the report ID, click the report and click the **Copy Shortcut** link. A window opens with a link such as:

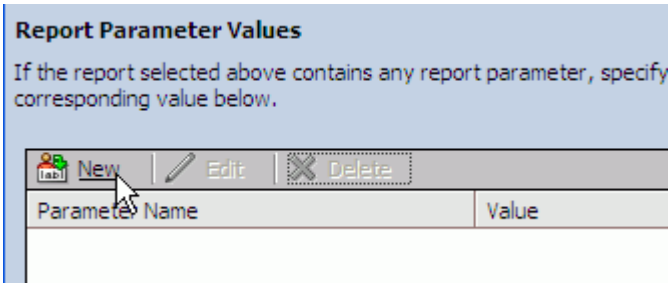
  ```
  http://myServer/oraclebam/ReportServer/default.aspx?Event=ViewReport&
  ReportDef=1&Buttons=False
  ```

  In this link the **ReportDef** value, `1`, is the report ID of the report `Emp_Report`. Every report in Oracle Business Activity Monitoring has a unique report ID.

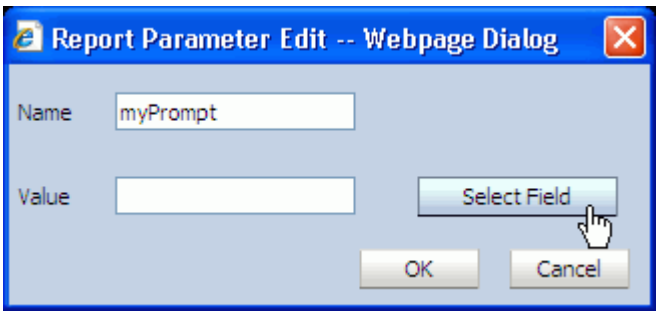
3. Configure the Report Parameter Values.

   Enter all of the parameters required by the report.

   Click **New** in the **Report Parameter Values** list to configure the parameter.

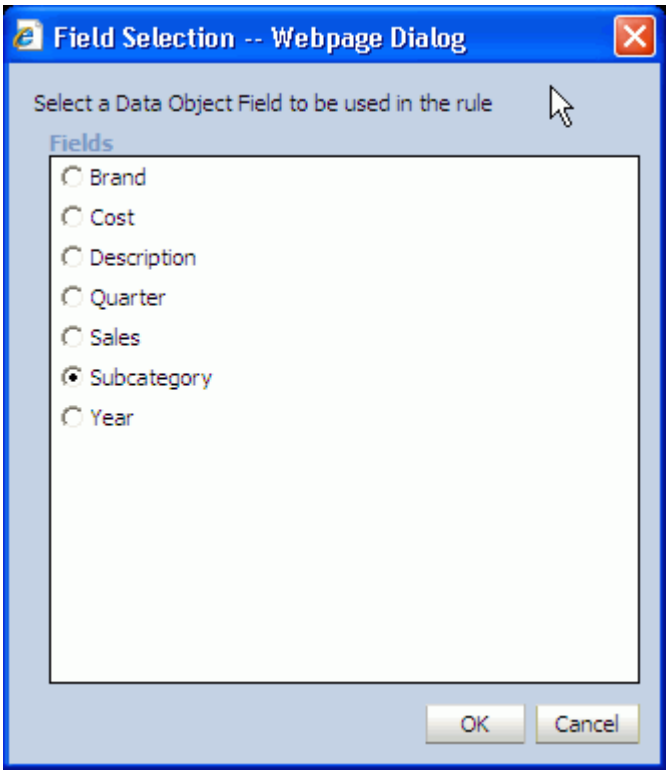


   Enter the parameter name in the **Name** field, and click Select Field to select the field on which the parameter acts.

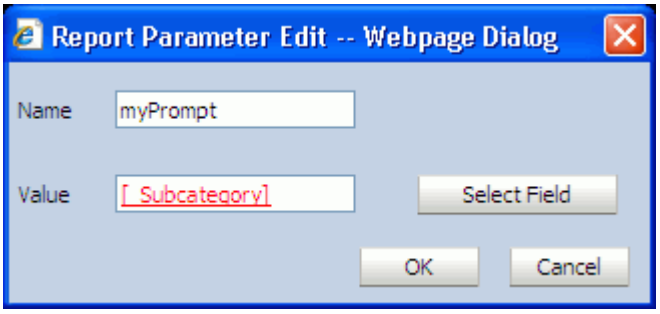


   Key in the parameter value, or select the field from the **Field Selection** dialog box, and click **OK**.

   For special values use the underscore (_), for example, _ALL_, _BLANK_, and _NULL_.



The selected field ID appears in the **Value** text box. Click **OK** to confirm and return to the parameters list.



F.3.5 Send a parameterized message for every matching row in a data object

This action can pick up recipients, message content, and the message subject from rows of a static data object. You can specify filter conditions in the configuration screen to choose data object rows for conditional notification. This action can be configured with any event, condition, or action.

When this action is invoked, the rows in the data object that match the filter criteria are used to construct an e-mail message (using the data object parameters specified in the action) which is sent out to the recipients. Message creation is similar to that in Section F.3.4, "Send a parameterized message."

To configure the action:

1. Click the **this data field has a condition of X** link and select the data object and filter condition to select the desired rows.

Note: The data object and filter conditions must be selected first so that the data object fields appear.

2. Click the **create message** link, and compose the e-mail message, with data object fields if required, similar to Section F.3.4, "Send a parameterized message".
3. Click the **set parameters** link and select a recipient (in **User**) and a report (optional), similar to Section F.3.4, "Send a parameterized message".
4. Click **OK** to save the rule.

F.3.6 Launch a rule

Select a dependent rule that includes the when this rule is launched event. For an example of constructing a dependent rule see Section 57.5, "Creating Complex Alerts."

F.3.7 Launch rule if an action fails

Select a dependent rule to launch if any of the actions included in the rule fail. For an example of constructing a dependent rule see Section 57.5, "Creating Complex Alerts"

F.3.8 Delete rows from a Data Object

Select the data object, and construct a filter entry such that when the filter condition is met the row is removed from the data object.

If the data being deleted is more than 10,000 rows, be aware of the following items:

- If any reports that are dependent upon the data object from which data is being deleted are open at the time the **Delete rows from a Data Object** action executes, the active data is stopped on the viewsets and reloaded after deletion is complete. Also, if a user attempts to open a report while the delete action for a dependent data object is in process, the report gets stuck or the outcome may be undefined. It is recommended that users do not open reports dependent on the data object while this action is in process. The reports continue to receive active data when the action is finished.
- In addition, during **Delete rows from a Data Object** execution, any alerts that are dependent on that data object are temporarily disabled internally. While this action is being run, any new alert created using that data object, or any dependent existing alerts that are disabled and reenabled, results in the system getting stuck. It is recommended that users do not create, disable, or reenable any alerts dependent on the data object while this action is in process. The alerts continue to function normally after the action is finished.

F.3.9 Call a Web Service

When this action is selected, do the following steps to configure the web service:

1. Enter the web service or WSIL endpoint URL. The URL must begin with the http scheme and must end in a valid extension (?WSDL, .WSDL or .WSIL).

   For example:

   ```
   http://host_name:port_
   number/OracleBAMWS/WebServices/DataObjectOperationsByID?WSDL

   http://api.google.com/GoogleSearch.wsdl
   ```

`http://host_name:port_number/inspection.wsil`

If it is a secure web service select the box and enter the required credentials.

Note: Oracle BAM cannot determine if the web service is hosted on a server which is behind a secure server. It is your responsibility to indicate whether the web service is behind an HTTP basic authentication based server, and you must enter valid credentials if they are required.

To accomplish one-way SSL, the Alert Web service client must be pointed to a trust store in which it can look up, to determine if the certificate presented to it exists in it or not. This can be done by setting properties in `BAMCommonConfig.xml`. See "Calling Secure Web Services" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

---

2. If it is a secure web service check the **Secure Web Service** checkbox and enter the required credentials.
3. Click **Display Services** to display the available services of the URL entered in the field.
4. The **Endpoint URL** field, which is initially disabled and empty, is populated after you enter the WSDL/WSIL and credentials, get the list of operations, and select an operation.

   It is populated with the endpoint URL defined in the WSDL file of the web service. If you find this endpoint URL outdated (for example, the web service implementation moved to a different endpoint but you do not have the new WSDL, but know the new endpoint URL) or incorrect, or want to override it, you can edit this URL. When the web service is invoked by Oracle BAM Event Engine, the configured endpoint URL is used to invoke the web service.
5. Click **Map Parameters**.

   When the event is based on a data object change (for example, When a data field changes in data object, When a data field in a report meets specified conditions, When a data field in a data object meets specified conditions), a selection list of fields to which the parameter can be mapped is displayed.

   To map the parameters choose the **Data Object Field** option, and select a data object field from the list next to each web service parameter listed in the Alert Web Service - Parameter Mapping dialog box.

   When the event is not based on a data object change, the value is entered in a text box.
6. Click **OK** to close the Alert Web Service - Parameter Mapping dialog box and the Alert Web Service Configuration dialog box.

See Section F.3.9.1, "How to Use Call a Web Service: An Example" for a specific example.

Note: If the web service does not respond to the call, then there are no logs available pertaining to the non-response or failure.

F.3.9.1 How to Use Call a Web Service: An Example

The following procedure details the steps to create a alert which invokes a web service, using the sample Employees data object to insert a row in a data object.

To use Call a Web Service:

1. Ensure that the /Samples/Employees data object exists in your Oracle BAM instance.
2. Log in to Oracle BAM web applications, and open Oracle BAM Active Studio.
3. Select the **Alerts** tab, and click **Create a New Alert**.
4. Click **Create a Rule**.
5. In the **Select an Event** list, select the first option: **In a specific amount of time**.
6. Click **select time interval** in the **Rule Expression** panel, and select **1 Second** as the time.
7. Click **OK** and **Next**.
8. In the **Select an Action** list, select the action **Call a Web Service**.
9. Click **configure web service** in the **Rule Expression** panel.

   The Alert Web Service Configuration dialog opens.
10. Provide the WSDL of the DataObjectOperationByName web service on your instance. The URL looks like:

    ```
    http://host_name:port_number/OracleBAMWS/WebServices/
    DataObjectOperationsByName?WSDL
    ```

    where *host_name* and *port_number* are substituted with your Oracle BAM instance's host name and port number.
11. Select the **Secure Web Service** checkbox, and provide the credentials.
12. Click **Display Operations**, and in the operations listed, select the operation **Insert**.

    This populates the endpoint URL of your web service. If the endpoint your of your web service has changed, or you want to override it with some other implementation, provide the new endpoint URL, otherwise, leave it as it is.
13. Click **Map Parameters** to provide the values that map to the parameters in this web service.

    The web service operation in this example requires a value for only one parameter, an XML payload containing the row to insert in the data object.

    Enter the following text in the `xmlpayload` value and click.

    ```
    <DataObject Name="Employees" Path="/Samples"><Contents><Row><Column
     Name="Salesperson" Value="Greg Guan Gan" /><Column Name="Sales Area"
     Value="Northeast" /><Column Name="Sales Number" Value="1234"
     /></Row></Contents></DataObject>
    ```
14. Click **OK** to close the Alert Web Service Configuration dialog, and click **OK** in the Rule Creation and Edit dialog.
15. After one second, open Oracle BAM Architect and check the contents of the /Samples/Employees data object to verify that the new row with **Salesperson** name **Greg Guan Gan** is inserted in the data object.

F.3.10 Run an Oracle Data Integrator Scenario

Use this action to trigger a scenario in Oracle Data Integrator. This action is only available if the integration files for Oracle Data Integrator have been installed. See "Installing the Oracle Data Integrator Integration Files" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for more information.

Ensure that the Oracle Data Integrator agent is running and that the agent host, port, and login credentials are properly configured in Oracle Enterprise Manager Fusion Middleware Control. Oracle BAM cannot verify that the Oracle Data Integrator agent is running, and if it is not running, the alert fires, but the action is not carried out as expected. Also, Oracle BAM alerts that trigger Oracle Data Integrator scenarios do not track the success or failure of the Oracle Data Integrator scenario call, and it is not logged on the Oracle BAM side. See "Configuring Oracle Data Integrator Properties," in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

In the alert creation dialog box, select the Oracle Data Integrator scenario to invoke by selecting the scenario name and version from the dropdown list.

If the scenario uses variables in it, choose the values (type in a value or choose a field value from the data object) to pass to Scenario Variables in the same screen.

F.3.11 Call an External Action

Call an External Action is used to develop a custom action. For users whose requirements cannot be fulfilled by the actions provided by Oracle BAM, this feature is used to extend the action set.

See Section 57.8, "Calling an External Action" for details on how to configure this action.

F.4 Frequency Constraint

The Frequency Constraint feature prevents a user's email inbox from being flooded with alerts by limiting the number of alert messages that can be sent out during a given time interval.

Frequency Constraint can be edited only if it is appropriate for the event selected. otherwise it is disabled. It can be set to a value of time which could be in seconds, minutes, or hours.

This limits the number of times the rule launches in a given time period. With real-time data, transactions can occur every millisecond, so alerting frequency must be controlled.

G

Oracle BAM ICommand Operations and File Formats

This appendix provides a detailed reference for each operation and parameter available in the ICommand command-line utility and web service.

This appendix includes the following sections:

- Section G.1, "Summary of Individual Operations"
- Section G.2, "Detailed Operation Descriptions"
- Section G.3, "Format of Command File"
- Section G.4, "Format of Log File"
- Section G.5, "Sample Export File"
- Section G.6, "Regular Expressions"

For more information about ICommand see the following topics:

- Chapter 58, "Using ICommand"
- Section 56.5, "Using the ICommand Web Service"

G.1 Summary of Individual Operations

This section summarizes the parameters that can be used with each ICommand operation. You can also see a summary of these operations in the command window by entering `icommand` (without any parameters) at the command prompt.

Table G–1 summarizes the commands available in ICommand.

Table G–1 ICommand Command Summary

Command	Parameters
`clear`	`-name` *`itemname`* `[-type [dataobject|folder|distributionlist]]` For more information about `clear` see Section G.2.1, "Clear."

Table G–1 (Cont.) ICommand Command Summary

Command	Parameters
`delete`	[`-name` *`itemname`*] [`-type [dataobject|folder|report|rule|securityfilters|distributionlist|ems|eds|all]`] [`-match` *`pattern`*] [`-regex` *`regularexpression`*] [`-all [0|1]`] [`-systemobjects [0|1]`] For more information about `delete` see Section G.2.2, "Delete."
`export`	`-file` *`file_name`* [`-name` *`itemname`*] [`-type [dataobject|folder|report|rule|securityfilters|distributionlist|ems|eds|all]`] [`-match` *`pattern`*] [`-regex` *`regularexpression`*] [`-all [0|1]`] [`-systemobjects [0|1]`] [`-dependencies [0|1]`] [`-layout [0|1]`] [`-contents [0|1]`] [`-permissions [0|1]`] [`-owner [0|1]`] [`-header [0|1]`] [`-footer [0|1]`] [`-append [0|1]`] [`-preview [0|1]`] For more information about `export` see Section G.2.3, "Export."

`import`	`-file` *`file_name`*
	`-continueonerror`
	[`-delay` *`milliseconds`*]
	[`-updatelayout`]
	[`-mode [preserveid|update|overwrite|append|rename|error]`]
	[`-preserveowner`]
	[`-setcol` *`col_name`*`/[null|now|value:`*`override_value`*`]`]
	`[-preview]`
	For more information about `import` see Section G.2.4, "Import."

`rename`	`-name` *`itemname`* `-newname` *`newitemname`* [`-type [dataobject|folder|report|rule|`distributionlist`|`ems`|eds]`] For more information about `rename` see Section G.2.5, "Rename."

G.2 Detailed Operation Descriptions

This section details each of the ICommand commands, their parameters, and gives examples. It includes the following topics:

G.2.1 Clear

Clears the contents of an item in the Active Data Cache.

What it means to be *cleared* depends upon the item type:

- For Data Objects, all existing rows within the Data Object are deleted.
- For Folders, all contents of the Folder are deleted.
- For Distribution Lists, all members (users and groups) are removed from the distribution list.

Table G–2 Clear Command Parameters

Parameter	Description
`-name` *`itemname`*	The name of the item to be cleared. Required.
`-type` *`itemtype`*	The type of the item to be cleared. The following are valid: ■ `dataobject` (see Example G–1) ■ `folder` ■ `distributionlist` `dataobject` is assumed if this parameter is omitted.

Example G–1 Clearing a Data Object

```
icommand -cmd clear -name "/Samples/Call Center" -type dataobject
```

G.2.2 Delete

Deletes an item from the Active Data Cache.

Table G–3 Delete Command Parameters

Parameter	Description
`-all [0|1]`	Controls whether all items of the specified type are deleted (see Example G–5). A nonzero or omitted value means delete all items of the specified type, a zero (0) value means only delete the named (or matched) items. Zero is assumed if this parameter is omitted.
`-match` *`pattern`*	A DOS-style pattern matching string, using the asterisk (*) and question mark (?) characters. The items whose names match the pattern are deleted.
`-name` *`itemname`*	The name of the item to be deleted.

Table G–3 (Cont.) Delete Command Parameters

Parameter	Description
`-regex` *`regularexpr`*	A regular expression pattern matching string. The items whose names match the pattern are deleted. See Section G.6, "Regular Expressions" for more information.
`-systemobjects [0|1]`	Controls whether data objects in the `System` folder are included when the `all`, `match`, or `regex` parameters are used. Zero (0) means these data objects are not included. Zero is assumed if this parameter is omitted.
`-type` *`itemtype`*	The type of the item to be deleted. The following are valid: ▪ `dataobject` (see Example G–2) ▪ `folder` ▪ `report` (see Example G–5) ▪ `rule` ▪ `securityfilters` (For the specified Data Objects) ▪ `distributionlist` ▪ `ems` (Enterprise Message Source) ▪ `eds` (External Data Source) ▪ `all` (see Example G–6) `dataobject` is assumed if this parameter is omitted.

Example G–2 Deleting a Data Object

This command deletes a data object named `TestDO`. Note that **dataobject** type is assumed if the **type** parameter is not specified.

```
icommand -cmd delete -name TestDO
```

Example G–3 Deleting an Alert Rule

For any ICommand operation on alerts, the value of the `type` parameter is `rule`. This command deletes a rule named `MyAlert`.

```
icommand -cmd delete -type rule -name "MyAlert"
```

Example G–4 Deleting security filter defined on a data object

To delete security filters defined on a data object, the name of the data object must be specified, instead of name of the security filter. This command deletes all security filters defined on the data object `MyDataObject`.

```
icommand -cmd delete -type securityfilters -name "MyDataObject"
```

Example G–5 Deleting All Reports

This command deletes all objects of type report.

```
icommand -cmd delete -type report -all 1
```

Example G–6 Deleting All Objects

This command deletes all items except systemobjects (data objects in the `System` folder).

```
icommand -cmd delete -type all
```

G.2.3 Export

Exports information about one or more objects in the Active Data Cache to an XML file. See Section G.5, "Sample Export File" for an example of an exported data object.

Table G–4 Export Command Parameters

Parameter	Description
`-all [0|1]`	Controls whether all items of the specified type are exported. A nonzero or omitted value means export all items of the specified type, a zero value means only export the named (or matched) items. Zero (0) is assumed if this parameter is omitted. For Reports, Folders, and Rules, only the items owned by the user running ICommand are exported, unless the user running ICommand is an administrator. When an administrator runs ICommand, any user's items may be exported. See Example G–14, "Exporting All of the Reports in the System"
`-append [0|1]`	Controls whether the exported information is appended to any existing file. A nonzero value means append. Zero (0) means overwrite the contents of any existing files. Zero is assumed if this parameter is omitted, or if the value is omitted. The Append parameter must be used with the Header and Footer parameters as described in Example G–22, "Using Append Parameter in Export". When the Append parameter is used, the Header and Footer parameters must be defined. If they are not, ICommand includes XML header information and closing XML </OracleBAMExport> tags after each append to the export file. The file is unusable for importing into Oracle BAM, because the import stops when it finds the first </OracleBAMExport> closing tag and ignores the rest of the objects.
`-contents [0|1]`	Applies only to Data Objects. Controls whether content information (row, column values) is to be exported. A nonzero value means export content information. Zero (0) means do not export content information. nonzero is assumed if this parameter is omitted, or if the value is omitted.
`-dependencies [0|1]`	Applies to only to Data Objects. Controls whether other Data Objects that the exported Data Objects depend on in the lookup columns are exported. A nonzero value or the parameter present with no value specifies that if the Data Objects being exported contain lookup columns, then the Data Objects that are looked up are exported. Zero is assumed if this parameter is omitted, or if the value is omitted.
`-file` *`file_name`*	The name of the file to export to. Required. If the file does not exist, it is created. If the file does exist, any contents are overwritten, unless the `append` parameter is used. Because the file contains XML, it usually has an XML extension.

Table G–4 (Cont.) Export Command Parameters

Parameter	Description
`-footer [0|1]`	Controls whether closing XML information is written to the end of the export file. This can allow successive executions of ICommand to assemble one XML file by repeatedly appending to the same file. A nonzero value means write the closing information. Zero (0) means do not write the closing information. nonzero is assumed if this parameter is omitted, or if the value is omitted. When used with the Append parameter, you must set the Footer value appropriately, or the file cannot be used with ICommand Import. If Footer is not defined, each append includes closing </OracleBAMExport> tags and the import stops when the first closing tag is read and does not import the remaining objects defined in the file. See Example G–22, "Using Append Parameter in Export" for a sample using this parameter.
`-header [0|1]`	Controls whether XML header information is written to the front of the export file. This can allow successive executions of ICommand to assemble one XML file by repeatedly appending to the same file. A nonzero value means write the header. Zero(0) means do not write the header. nonzero is assumed if this parameter is omitted, or if the value is omitted. See Example G–22, "Using Append Parameter in Export" for a sample using this parameter.
`-layout [0|1]`	Applies only to Data Objects. Controls whether layout information is to be exported. A nonzero value means export layout information. Zero (0) means do not export layout information. nonzero is assumed if this parameter is omitted, or if the value is omitted.
`-match` *`pattern`*	A DOS-style pattern matching string, using the asterisk (*) and question mark (?) characters. The items whose names match the pattern are exported (see Example G–21, "Exporting a Data Object Using the Match Parameter").
`-name` *`itemname`*	The name of the item to be exported.
`-owner [0|1]`	Applies only to Folders, Reports, and Rules. Controls whether the information about the owner of the items being exported is included in the export. A nonzero value means export the owner information. Zero (0) means do not export the owner information. nonzero is assumed if this parameter is omitted, or if the value is omitted.
`-permissions [0|1]`	Applies only to Data Objects and Folders. Controls whether permissions information is to be exported. A nonzero value means export information about the permission settings of the exported Data Objects or Folders. Zero (0) means do not export permission information. Zero is assumed if this parameter is omitted, or if the value is omitted. For Data Objects, only the permissions of the Data Object itself are exported. Any permissions that might be on the folders or subfolders that the Data Objects are contained within are not included. For Folders, the permissions reflect the cumulative permissions of all parent Folders of the Folders being exported.

Table G–4 (Cont.) Export Command Parameters

Parameter	Description
`-preview [0|1]`	In `preview` mode, ICommand goes through the motions of exporting all of the specified items, but does not actually output any information. This can see what would be exported for a given command line, and what errors might occur. In this mode, ICommand export continues processing even after some errors that would cause non-preview mode to stop the export. A nonzero value means preview mode. nonzero is assumed if the value is omitted. Zero (0) is assumed if the parameter is omitted.
`-regex` *`regularexpr`*	A regular expression pattern matching string. The items whose names match the pattern are exported. See Section G.6, "Regular Expressions" for more information.
`-systemobjects [0|1]`	Controls whether Data Objects in the System folder are included when the `all`, `match`, or `regex` parameters are used. Zero (0) means these data objects are not included. Zero is assumed if this parameter is omitted.
`-type` *`itemtype`*	The type of the item to be exported. The following are valid: ▪ `dataobject` (see Example G–7 and Example G–8) ▪ `folder` (see Example G–9, Example G–10, and Example G–11) ▪ `report` (see Example G–12, Example G–13, and Example G–14) ▪ `rule` (see Example G–15) ▪ `securityfilters` (For the specified Data Objects) (see Example G–16) ▪ `distributionlist` (see Example G–17) ▪ `ems` (Enterprise Message Source) (see Example G–18) ▪ `eds` (External Data Source) (see Example G–19) ▪ `all` (see Example G–20) `dataobject` is assumed if this parameter is omitted.

Example G–7 Exporting a Data Object in a Folder

```
icommand -cmd export -name "/Samples/Call Center" -file "C:\CallCenter.xml"
```

Note that the `type` parameter was not included in this example. By default `dataobject` is assigned to `type` if it is not specified.

Example G–8 Exporting a Data Object at the Root

```
icommand -cmd export -name TestDataObject -file "C:\TestDataObject.xml"
```

Note that the data object name was not preceded by the slash (/). When a Data Object is in the root Data Objects folder, a slash is not required.

Example G–9 Exporting a Folder from My Reports

In the first case, the `private:`*`owner`*`/Report` prefix is used in the `name` parameter because the user exporting the folder is not the folder owner.

```
icommand -cmd export -name "/private:bamadmin/Report/TestMainFolder/TestSubFolder"
 -type folder -file C:\FolderExportTest.xml
```

In the second case, the `private:owner/Report` prefix was not used in the `name` parameter because the user exporting the folder is the folder owner.

```
icommand -cmd export -name "/TestMainFolder/TestSubFolder" -type folder -file
 C:\FolderExportTest.xml
```

Example G–10 Exporting a Folder from Shared Reports

```
icommand -cmd export -name "/public/Report/MainFolderInShared" -type folder -file
 C:\FolderExportTest2.xml
```

Note that the `public` prefix is added to the `name` parameter.

Example G–11 Exporting a Folder from Data Objects

```
icommand -cmd export -name "/public/DataObject/Test Sub folder" -type folder -file
 C:\foldertest1.xml
```

Example G–12 Exporting a Private Report

As in Example G–9, there are two methods of exporting private reports.

```
icommand -cmd export -name "/private:bamadmin/Report/MyReport" -type report -file
C:\MyReport.xml

icommand -cmd export -name MyReport -type report -file C:\MyReport.xml
```

Example G–13 Exporting a Shared Report

```
icommand -cmd export -name "/public/Report/SharedReport" -type report -file
C:\SharedReport.xml
```

Example G–14 Exporting All of the Reports in the System

```
icommand -cmd export -type report -all -file C:\temp\TestAll.xml
```

Example G–15 Exporting an Alert Rule

```
icommand -cmd export -name Alert1 -type rule -file C:\Alert1.xml
```

Example G–16 Exporting a Security Filter

```
icommand -cmd export -type securityfilters -name "TestDO" -file
"C:\TestFilter.xml"
```

Note that in the `name` parameter the name of the Data Object is specified rather than the name of the security filter.

Example G–17 Exporting a Distribution List

```
icommand -cmd export -name MyDistList -type distributionlist -file
C:\MyDistList.xml
```

Example G–18 Exporting an Enterprise Message Source

```
icommand -cmd export -type ems -name TestEMS -file C:\TestEMS.xml
```

Example G–19 Exporting an External Data Source

```
icommand -cmd export -type eds -name TestEDS -file C:\TestEDS.xml
```

Example G–20 Exporting All Oracle BAM Objects in the System

```
icommand -cmd export -type all -file C:\temp\TestAll.xml
```

Example G–21 Exporting a Data Object Using the Match Parameter

```
icommand -cmd export -match "/M*" -file "c:/exportDOstartingwithM.xml"
```

Example G–22 Using Append Parameter in Export

In the first case (the incorrect example), Append is used without setting the Header and Footer parameters (by default Header and Footer are set to 1).

```
icommand -cmd export -type dataobject -name "/Samples/Call Center" -file do.xml
icommand -cmd export -type dataobject -name "/Samples/Employees" -file do.xml
-append
icommand -cmd export -type dataobject -name "/Samples/Film Sales" -file do.xml
-append
```

The output from these commands is as follows. Notice that an XML header and closing tags are included with each append to the file. If this file is used for importing data into Oracle BAM, only the first object is imported. As soon as the first </OracleBAMExport> is read at line 4, the import stops.

```
<?xml version="1.0"?>
<OracleBAMExport Version="2020">
  <exported object/>
</OracleBAMExport>
<?xml version="1.0"?>
<OracleBAMExport Version="2020">
  <exported object/>
</OracleBAMExport>
<?xml version="1.0"?>
<OracleBAMExport Version="2020">
  <exported object/>
</OracleBAMExport>
```

In the second case (the correct example), The Header and Footer parameters are specified to produce the necessary output.

```
icommand -cmd export -type dataobject -name "/Samples/Call Center" -file do2.xml
 -header 1 -footer 0
 //only the footer is supressed in the first command
icommand -cmd export -type dataobject -name "/Samples/Employees" -file do2.xml
 -append 1 -header 0 -footer 0
 //both the header and the footer are suppressed in the intermediate commands
icommand -cmd export -type dataobject -name "/Samples/Film Sales" -file do2.xml
 -append 1 -header 0 -footer 1
 //only the header is suppressed in the last commands
```

The output file produced by these commands can import the objects into an Oracle BAM Server.

```
<?xml version="1.0"?>
<OracleBAMExport Version="2020">
  <exported object>
  <exported object>
</OracleBAMExport>
```

G.2.4 Import

Imports the information from an XML file to an object in the Active Data Cache. The object may be created, replaced, or updated.

If the object does not exist, it is created if possible. For Data Objects, the input file must contain layout information to create the Data Object, and if the file contains no content information, then an empty Data Object is created.

If the user running ICommand is not an administrator, Reports are always imported to the private folders of the user running ICommand. If the path information in the import file exactly matches existing private folders of the user running ICommand, the imported report is placed in that location. Otherwise, it is placed into the root of that user's private folders.

If the user running ICommand is an administrator, then the `preserveowner` option may be used to allow Folders, Reports and Rules to be imported with their original ownership and to their original location.

Table G–5 Import Command Parameters

Parameter	Description
`-continueonerror [0|1]`	While importing objects from a file, by default, ICommand stops whenever an error is encountered. If you are importing several objects and do not want to stop when an error is found in one, use the continueonerror parameter to continue importing the rest of the objects specified in the command. Specify a one (1) to ignore errors and continue importing other objects (see Example G–23).
`-delay` *`millisec`*	Applies only to Data Objects. A value that specifies a delay that is to occur between each row insertion or update. This can simulate active data at a specified rate. The number is the number of milliseconds to wait between each row. It must be greater than zero. If this parameter is omitted, there is no delay. See Example G–23, "Importing a Data Object With Delay"
`-file` *`file_name`*	The name of the file to import from. Required. This would usually be a file that was created through the export command.
`-preserveowner`	Applies only to Folders, Reports, and Rules. Controls whether, when the item is imported, the ownership of the item is set as specified in the import file. This setting of ownership can only be done if the ownership was included in the file during export, and if the user running ICommand is an administrator. A nonzero value means set the ownership as specified in the import file. Zero (0) means the imported items remain owned by the user running ICommand. Zero is assumed if this parameter is omitted, or if the value is omitted.

Table G–5 (Cont.) Import Command Parameters

Parameter	Description
`-preview [0|1]`	In `preview` mode, ICommand goes through the motions of importing all of the specified items, but does not actually input any information. This can see what would be imported for a given command line, and what errors might occur. In this mode, ICommand import continues processing even after some errors that would cause non-preview mode to stop the import. A nonzero value means preview mode. nonzero is assumed if the value is omitted. Zero (0) is assumed if the parameter is omitted. This parameter is supported for the following objects: Rule, Distribution list, EDS, EMS, Report, Folder, and Security Filters. See Example G–24, "Importing a Report in Preview mode"

Table G–5 (Cont.) Import Command Parameters

Parameter	Description
`-mode` *mode*	By default, if the mode parameter is not specified, the value Error is assumed for objects of type Folder, Report, EDS, EMS, and Distribution List. The following mode values are valid for Folders, Reports, EMS, and EDS objects: ■ `overwrite` If the item exists, replaces it with the imported item. ■ `rename` If the item exists, changes the name of the imported item. The new name is computed automatically and reported in a message. ■ `error` If the item exists, terminates the import with an error. The following values are valid for Distribution List objects: ■ `overwrite` If the item exists, replaces it with the imported item. ■ `rename` If the item exists, changes the name of the imported item. The new name is computed automatically and reported in a message. ■ `append` If the item exists, appends the users in the imported list to the existing list. ■ `error` If the item exists, terminates the import with an error. The following value is supported for Data Objects or Reports: ■ `preserveid` This option is important because some other items, such as Reports, point to the Data Objects they use by ID, not by name. Data Object Usage: If the imported Data Object does not exist and must be created, ICommand attempts to assign the Data Object the same internal ID that the exported Data Object had. If it cannot, the import is terminated with an error. Report Usage: If the imported Report does not exist and must be created, ICommand attempts to assign the Report the same internal ID that the exported Report had. If it cannot, the import is terminated with an error.

Table G–5 (Cont.) Import Command Parameters

Parameter	Description
`-mode` *`mode`* (cont.)	Only the following value is valid for Data Objects: ▪ `update` Typically, when ICommand imports a Data Object, it creates a new Data Object or locates the existing Data Object and inserts the imported rows into that Data Object. In `update` mode, ICommand instead attempts to locate existing matching rows by Row ID, and updates those existing rows with the values in the import file. Unmatched rows are inserted. For matching Row IDs in the import file that have no data columns specified, the rows are deleted from the existing Data Object. For Security Filters, the only value supported is `overwrite`. If `overwrite` is not specified and the Data Object contains at least one Security Filter, the import is terminated with an error. This parameter is not supported for Rules.

`-setcol`	Allows override of column values from the command line during import, including setting to current date/time.
	`-setcol` *`column_name`*`/NULL`
	`-setcol` *`column_name`*`/NOW`
	`-setcol` *`column_name`*`/VALUE:`*`override-value`*
	`column_name` is the name of a column in the Data Object being imported. This cannot be a column of type lookup or calculated. Column names that are not contained in the input XML being imported can be specified, if they are columns in the Data Object being imported into.
	The portion after the slash specifies a value that should be substituted for that column on each row that is imported -- any value for that column in the import file is ignored (overridden). Note that slash is the one character that is not permitted in column names, so there is no potential conflict with any column names in this syntax.
	`NULL` specifies that the column value should be set to null. The column must be defined as "nullable" in the Data Object's layout.
	`NOW` specifies that the column value should be set to the current date/time when the column value is being set into the row. This option can only be used for columns of type datetime, timestamp, and string.
	`VALUE:`*`override-value`* specifies an arbitrary constant value (after the colon) that the column should be set to. The value must be a legal value for the type of the column.
	To allow multiple columns to be overridden, any number of `setcol` parameters may be present. However, because duplicate parameters are not permitted, ICommand recognizes any parameter name that starts with `setcol` as a `setcol` parameter (for example, `setcol1`, `setcol2`, and so on).
	Sample command line:
	`icommand -cmd import -file myfile.xml -setcol1 Field1/null -setcol2 Field3/now -setcol3 "Customer Name/value:John Q. Public"`

Table G–5 (Cont.) Import Command Parameters

Parameter	Description
`-updatelayout`	Applies only to Data Objects. Controls whether, if the Data Object being imported exists, the layout (schema) of the Data Object is updated according to the layout information in the import file. True if parameter is present; false if parameter is not present.

Example G–23 Importing a Data Object With Delay

```
icommand -cmd import -file C:\TestDO.xml -delay 1000 -continueonerror 1
```

Example G–24 Importing a Report in Preview mode

```
icommand -cmd import -file C:\TestReport.xml -preview 1
```

G.2.5 Rename

Renames an item in the Active Data Cache.

Table G–6 Rename Command Parameters

Parameter	Description
`-name` *`itemname`*	The name of the item to be renamed. Required. The full folder path must be given when renaming objects of type Folder (see Example G–26, "Renaming Folders").
`-newname` *`newitemname`*	The new name for the item. Required. The full folder path must be given when renaming objects of type Folder (see Example G–26, "Renaming Folders"). For Data Objects and Reports, only the new base name should be given, with no path (for example -newname "MyReport").
`-type` *`itemtype`*	The type of object to be renamed. The following are valid: ▪ `dataobject` (see Example G–25) ▪ `folder` (see Example G–26) ▪ `report` (see Example G–27) ▪ `rule` ▪ `distributionlist` (see Example G–28) ▪ `ems` (Enterprise Message Source) ▪ `eds` (External Data Source) `dataobject` is assumed if this parameter is omitted. `all` is not supported as an item type in the `rename` command.

Example G–25 Renaming a Data Object in a Folder

```
icommand -cmd rename -type dataobject -name "/TestDataObjectFolder/TestDataObject"
 -newname NewTestDataObject
```

Example G–26 Renaming Folders

Renaming a data object folder:

```
icommand -cmd rename -type folder -name "/public/DataObject/TestFolder"
 -newname "/public/DataObject/NewTestFolder"
```

Renaming a private report folder:

```
icommand -cmd rename -type folder -name "/private:weblogic/Report/MySubFolder"
 -newname "/private:weblogic/Report/NewMySubFolder"
```

Renaming a shared report folder

```
icommand -cmd rename -type folder -name "/public/Report/TestSubFolder"
-newname "/public/Report/NewTestSubFolder"
```

Example G–27 Renaming a Report in a Private Folder

```
icommand -cmd rename -type report -name "/TestReportFolder/TestReport" -newname
 NewTestReport
```

Example G–28 Renaming a Distribution List

```
icommand -cmd rename -type distributionlist -name TestList -newname MyDistList
```

Example G–29 Renaming an Alert Rule

For any ICommand operation on alerts, the value of the `type` parameter is `rule`. This command renames a rule named `MyAlert`.

```
icommand -cmd rename -type rule -name "MyAlert" -newname "MyRenamedAlert"
```

G.3 Format of Command File

This section contains the following topics:

- Section G.3.1, "Inline Content"
- Section G.3.2, "Command IDs"
- Section G.3.3, "Continue On Error"

The command file contains the root tag `OracleBAMCommands`.

Within the root tag is a tag for every command to be executed. The tag name is the command name, and the parameters for the command are attributes.

Sample command file:

```
<?xml version="1.0" encoding="utf-8"?>
<OracleBAMCommands continueonerror="1">
  <Export name="Samples/Media Sales" file="MediaSales.xml" contents="0" />
  <Rename name="Samples/Call Center" newname="Call Centre" />
  <Delete type="EMS" name="WebLog" />
  <Delete type="EMS" name="WebLog2" />
</OracleBAMCommands>
```

The output of this sample command file is shown in Section G.4, "Format of Log File."

G.3.1 Inline Content

When using a command file to import, the `inline` option enables you to include the import content inside the command file, rather than in a separate import file. Here is an example:

```
<?xml version="1.0"?>
<OracleBAMCommands>
<Import inline="1">
<OracleBAMExport Version="2013">
```

```
  <DataObject Version="14" Name="Employees_Inline" ID="_Employees_Inline"
    Path="/Samples" External="0">
    <Layout>
      <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
        Nullable="1" Public="1"/>
      <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
        Nullable="1" Public="1"/>
      <Column Name="Sales Number" ID="_Sales_Number" Type="integer"
        Nullable="1" Public="1"/>
      <Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0"
        Public="1"/>
      <Indexes/>
    </Layout>
    <Contents>
      <Row ID="1">
        <Column ID="_Salesperson" Value="Greg Masters"/>
        <Column ID="_Sales_Area" Value="Northeast"/>
        <Column ID="_Sales_Number" Value="567"/>
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
      </Row>
      <Row ID="2">
        <Column ID="_Salesperson" Value="Lynette Jones"/>
        <Column ID="_Sales_Area" Value="Southwest"/>
        <Column ID="_Sales_Number" Value="228"/>
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
      </Row>
      <Row ID="3">
        <Column ID="_Salesperson" Value="Noel Rogers"/>
        <Column ID="_Sales_Area" Value="Northwest"/>
        <Column ID="_Sales_Number" Value="459"/>
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
      </Row>
    </Contents>
  </DataObject>
</OracleBAMExport>
</Import>
</OracleBAMCommands>
```

G.3.2 Command IDs

This feature is only used when output is being sent to a log file. To make the parsing of log results easier, each command can be given an ID. This ID is included in the Result or Error elements of any output related to that command.

Sample Input:

```
<OracleBAMCommands continueonerror="1">
  <Delete id="1" type="dataobject" name="Data Object A"/>
  <Delete id="2" type="dataobject" name="Data Object B"/>
</OracleBAMCommands>
```

Sample Output Log File:

```
<?xml version="1.0"?>
<ICommandLog Login="weblogic">
  <Results Command="Delete" ID="1">Data Object &amp;quot;/Data Object A&amp;quot;
 deleted.</Results>
  <Error Command="Delete" ID="2">
    <![CDATA[BAM-02409: There is no Data Object named "Data Object B".
```

```
    [ErrorSource="ICommandEngine",ErrorID="ICommandEngine.DOExist"]]]>
  </Error>
</ICommandLog>
```

G.3.3 Continue On Error

Ordinarily, ICommand executes commands in a command file until a failure occurs, or until they all complete successfully. In other words, if a command file contains 20 commands, and the second command fails for any reason, then no further commands are executed. This behavior can be changed by using the `continueonerror` attribute at either a global level or for each command.

Example G–30 shows how to use the `continueonerror` attribute so that all commands are executed regardless of if any failures occur

Example G–30 Enabling Global ContinueOnError Mode

```
<OracleBAMCommands  continueonerror="1">
  <Delete id="1" type="dataobject" name="Data Object A"/>
  <Delete id="2" type="dataobject" name="Data Object B"/>
</OracleBAMCommands>
```

In Example G–31, `continueonerror` only applies to the command that deletes Data Object A. If this command fails, then ICommand outputs the error and continues. But if any other command fails, ICommand stops immediately.

Example G–31 Enabling Command-Level ContinueOnError Mode

```
<OracleBAMCommands>
  <Delete id="1" type="dataobject" name="Data Object A" continueonerror="1"/>
  <Delete id="2" type="dataobject" name="Data Object B"/>
  <Delete id="3" type="dataobject" name="Data Object C"/>
  <Delete id="4" type="dataobject" name="Data Object D"/>
</OracleBAMCommands>
```

G.4 Format of Log File

The log file contains the root tag `ICommandLog`.

Within the root tag is an entry for every error or informational message logged.

Errors are logged with the tag `Error`.

Informational messages are logged with the tag `Results`.

Both `Results` and `Error` tags optionally contain an attribute of the form `Command=`*`cmdname`*, if appropriate, that contains the name of the command that generated the error or informational message.

This sample log file is output of command file given in Section G.3, "Format of Command File":

```
<?xml version="1.0" encoding="utf-8"?>
<ICommandLog Login="user_name">
  <Results Command="Export">Data Object "/Samples/Media Sales" exported
 successfully (0 rows).</Results>
  <Results Command="Export">1 items exported successfully.</Results>
  <Results Command="Rename">Data Object "/Samples/Call Center" renamed to
 "/Samples/Call Centre".</Results>
  <Results Command="Delete">Enterprise Message Source "WebLog" deleted.</Results>
  <Error Command="Delete"><![CDATA[Error while processing command "Delete".
```

```
 [ErrorSource="ICommand", ErrorID="ICommand.Error"] There is no Enterprise Message
 Source named "WebLog2". [ErrorSource="ICommand",
 ErrorID="ICommand.EMSExist"]]]></Error>
</ICommandLog>
```

G.5 Sample Export File

The following example shows a sample file resulting from exporting a Data Object.

```
<?xml version="1.0"?>
<OracleBAMExport Version="2018">
  <DataObject Version="14" Name="Employees" ID="_Employees" Path="/Samples"
 External="0">
    <Layout>
      <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
 Nullable="1" Public="1"/>
      <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
 Nullable="1" Public="1"/>
      <Column Name="Sales Number" ID="_Sales_Number" Type="integer" Nullable="1"
 Public="1"/>
      <Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0"
 Public="1"/>
      <Indexes/>
    </Layout>
    <Contents>
      <Row ID="1">
        <Column ID="_Salesperson" Value="Greg Masters"/>
        <Column ID="_Sales_Area" Value="Northeast"/>
        <Column ID="_Sales_Number" Value="567"/>
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
      </Row>
      <Row ID="2">
        <Column ID="_Salesperson" Value="Lynette Jones"/>
        <Column ID="_Sales_Area" Value="Southwest"/>
        <Column ID="_Sales_Number" Value="228"/>
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
      </Row>
      <Row ID="3">
        <Column ID="_Salesperson" Value="Noel Rogers"/>
        <Column ID="_Sales_Area" Value="Northwest"/>
        <Column ID="_Sales_Number" Value="459"/>
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
      </Row>
    </Contents>
  </DataObject>
</OracleBAMExport>
```

G.6 Regular Expressions

The `export` and `delete` commands optionally accept a regular expression with the `regex` parameter.

A regular expression is a pattern of text that consists of ordinary characters (for example, letters a through z) and special characters, known as *metacharacters*. The pattern describes one or more strings to match when searching for items by name.

Note: The behavior of ICommand -regex is exactly like the java.util.regex package for matching character sequences against patterns specified by regular expressions.

Table G–7 contains the complete list of metacharacters and their behavior in the context of regular expressions.

Table G–7 Metacharacters for Regular Expressions

Character	Description
\	Marks the next character as a special character, a literal, a backreference, or an octal escape. For example, 'n' matches the character "n". '\n' matches a newline character. The sequence '\\' matches "\" and "\(" matches "(".
^	Matches the position at the beginning of the input string. If the RegExp object's `Multiline` property is set, ^ also matches the position following '\n' or '\r'.
$	Matches the position at the end of the input string. If the RegExp object's `Multiline` property is set, $ also matches the position preceding '\n' or '\r'.
*	Matches the preceding character or subexpression zero or more times. For example, zo* matches "z" and "zoo". * is equivalent to {0,}.
+	Matches the preceding character or subexpression one or more times. For example, 'zo+' matches "zo" and "zoo", but not "z". + is equivalent to {1,}.
?	Matches the preceding character or subexpression zero or one time. For example, "do(es)?" matches the "do" in "do" or "does". ? is equivalent to {0,1}
{*n*}	*n* is a nonnegative integer. Matches exactly *n* times. For example, 'o{2}' does not match the 'o' in "Bob," but matches the two o's in "food".
{*n*,}	*n* is a nonnegative integer. Matches at least *n* times. For example, 'o{2,}' does not match the "o" in "Bob" and matches all the o's in "fooooood". 'o{1,}' is equivalent to 'o+'. 'o{0,}' is equivalent to 'o*'.
{*n*,*m*}	*M* and *n* are nonnegative integers, where *n* <= *m*. Matches at least *n* and at most *m* times. For example, "o{1,3}" matches the first three o's in "fooooood". 'o{0,1}' is equivalent to 'o?'. Note that you cannot put a space between the comma and the numbers.
?	When this character immediately follows any of the other quantifiers (*, +, ?, {*n*}, {*n*,}, {*n*,*m*}), the matching pattern is non-greedy. A non-greedy pattern matches as little of the searched string as possible, whereas the default greedy pattern matches as much of the searched string as possible. For example, in the string "oooo", 'o+?' matches a single "o", while 'o+' matches all 'o's.
.	Matches any single character except "\n". To match any character including the '\n', use a pattern such as '[\s\S]'.
(*pattern*)	A subexpression that matches *pattern* and captures the match. The captured match can be retrieved from the resulting Matches collection using the `$0...$9` properties. To match parentheses characters ( ), use '\(' or '\)'.

Table G–7 (Cont.) Metacharacters for Regular Expressions

Character	Description
(?:*pattern*)	A subexpression that matches *pattern* but does not capture the match, that is, it is a non-capturing match that is not stored for possible later use. This is useful for combining parts of a pattern with the "or" character (|). For example, 'industr(?:y | ies) is a more economical expression than 'industry | industries'.
(?=*pattern*)	A subexpression that performs a positive lookahead search, which matches the string at any point where a string matching *pattern* begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?=95 | 98 | NT | 2000)' matches "Windows" in "Windows 2000" but not "Windows" in "Windows 3.1". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
(?!*pattern*)	A subexpression that performs a negative lookahead search, which matches the search string at any point where a string not matching *pattern* begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?!95 | 98 | NT | 2000)' matches "Windows" in "Windows 3.1" but does not match "Windows" in "Windows 2000". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
x | *y*	Matches either *x* or *y*. For example, 'z | food' matches "z" or "food". '(z | f)ood' matches "zood" or "food".
[*xyz*]	A character set. Matches any of the enclosed characters. For example, '[abc]' matches the 'a' in "plain".
[^*xyz*]	A negative character set. Matches any character not enclosed. For example, '[^abc]' matches the 'p' in "plain".
[*a-z*]	A range of characters. Matches any character in the specified range. For example, '[a-z]' matches any lowercase alphabetic character in the range 'a' through 'z'.
[^*a-z*]	A negative range characters. Matches any character not in the specified range. For example, '[^a-z]' matches any character not in the range 'a' through 'z'.
\b	Matches a word boundary, that is, the position between a word and a space. For example, 'er\b' matches the 'er' in "never" but not the 'er' in "verb".
\B	Matches a nonword boundary. 'er\B' matches the 'er' in "verb" but not the 'er' in "never".
\c*x*	Matches the control character indicated by *x*. For example, \cM matches a Control-M or carriage return character. The value of *x* must be in the range of A-Z or a-z. If not, c is assumed to be a literal 'c' character.
\d	Matches a digit character. Equivalent to [0-9].
\D	Matches a nondigit character. Equivalent to [^0-9].
\f	Matches a form-feed character. Equivalent to \x0c and \cL.
\n	Matches a newline character. Equivalent to \x0a and \cJ.
\r	Matches a carriage return character. Equivalent to \x0d and \cM.

Table G–7 (Cont.) Metacharacters for Regular Expressions

Character	Description
\s	Matches any white space character including space, tab, form-feed, and so on. Equivalent to [\f\n\r\t\v].
\S	Matches any non-white space character. Equivalent to [^ \f\n\r\t\v].
\t	Matches a tab character. Equivalent to \x09 and \cI.
\v	Matches a vertical tab character. Equivalent to \x0b and \cK.
\w	Matches any word character including underscore. Equivalent to '[A-Za-z0-9_]'.
\W	Matches any nonword character. Equivalent to '[^A-Za-z0-9_]'.
\x*n*	Matches *n*, where *n* is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, '\x41' matches "A". '\x041' is equivalent to '\x04' & "1". Allows ASCII codes to be used in regular expressions.
num	Matches *num*, where *num* is a positive integer. A reference back to captured matches. For example, '(.)\1' matches two consecutive identical characters.
n	Identifies either an octal escape value or a backreference. If *n* is preceded by at least *n* captured subexpressions, *n* is a backreference. Otherwise, *n* is an octal escape value if *n* is an octal digit (0-7).
nm	Identifies either an octal escape value or a backreference. If *nm* is preceded by at least *nm* captured subexpressions, *nm* is a backreference. If *nm* is preceded by at least *n* captures, *n* is a backreference followed by literal *m*. If neither of the preceding conditions exists, *nm* matches octal escape value *nm* when *n* and *m* are octal digits (0-7).
nml	Matches octal escape value *nml* when *n* is an octal digit (0-3) and *m* and *l* are octal digits (0-7).
\u*n*	Matches *n*, where *n* is a Unicode character expressed as four hexadecimal digits. For example, \u00A9 matches the copyright symbol (©).

H

Normalized Message Properties

This appendix describes normalized message properties.

This appendix includes the following sections:

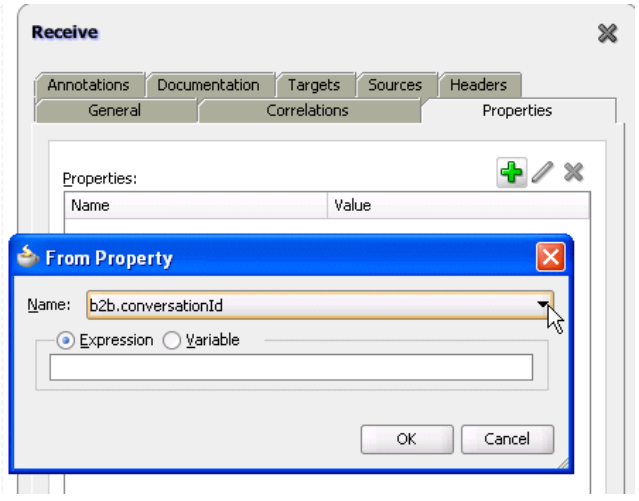
- Section H.1, "Introduction to Normalized Messages"
- Section H.2, "Oracle BPEL Process Manager Properties"
- Section H.3, "Oracle Web Services Addressing Properties"
- Section H.4, "Manipulating Normalized Message Properties with bpelx Extensions"

H.1 Introduction to Normalized Messages

Header manipulation and propagation is a key business integration messaging requirement. Oracle BPEL Process Manager, Oracle Mediator, Oracle JCA adapters, and Oracle B2B rely extensively on header support to solve customers' integration needs. For example, you can preserve a file name from the source directory to the target directory by propagating it through message headers. In Oracle BPEL Process Manager and Oracle Mediator, you can access, manipulate, and set headers with varying degrees of user interface support.

Figure H–1 provides details.

Figure H–1 Properties Tab for Normalized Messages Header Properties



A normalized message is simplified to have only two parts, properties and payload.

Typically, properties are name-value pairs of scalar types. To fit the existing complex headers into properties, properties are flattened into scalar types.

The user experience is simplified while manipulating headers in design time, because the complex properties are predetermined. In the Mediator Editor or Oracle BPEL Designer, you can manipulate the headers with some reserved key words.

However, this method does not address the properties that are dynamically generated based on your input. Based on your choice, the header definitions are defined. These definitions are not predetermined and therefore cannot be accounted for in the list of predetermined property definitions. You cannot design header manipulation of the dynamic properties before they are defined. To address this limitation, you must generate all the necessary services (composite entry points) and references. This restriction applies to services that are expected to generate dynamic properties. Once dynamic properties are generated, they must be stored for each composite. Only then can you manipulate the dynamic properties in the Mediator Editor or Oracle BPEL Designer.

For more information on normalized message properties, see *Oracle Fusion Middleware User's Guide for Technology Adapters* and *Oracle Fusion Middleware User's Guide for Oracle B2B*.

H.2 Oracle BPEL Process Manager Properties

Table H–1 lists all the predetermined properties of a normalized message for Oracle BPEL Process Manager.

Table H–1 Properties for Oracle BPEL Process Manager

Property Name	Propagatable (Yes/No)	Direction (Inbound /Outbound)	Data Type	Range of Valid Values	Description
`bpel.metadata`	Yes	Both	String	Any string, size limit: 1000	This contains extra information with which you want to associate the BPEL instance. Whatever was passed in is stored in the metadata column of the `cube_instance` table.
`bpel.priority`	Yes	Inbound	String that can be read into an integer	(1-10). 1 being the highest priority	Goes into the `cube_instance` priority column. Used by the system to prioritize.
`bpel.title`	No	Inbound	String	Any string, size limit: 100	Goes into the `title` column of `cube_instance` table.

Table H–1 (Cont.) Properties for Oracle BPEL Process Manager

Property Name	Propagatable (Yes/No)	Direction (Inbound /Outbound)	Data Type	Range of Valid Values	Description
`bpel.instanceIndex1`	No	Inbound	String	Any string, size limit: 100	This goes into the `ci_indexes` table (extra index for the `cube_instance`).
`bpel.instanceIndex2`	No	Inbound	String	Any string, size limit: 100	This goes into the `ci_indexes` table (extra index for `cube_instance`).
`bpel.instanceIndex3`	No	Inbound	String	Any string, size limit: 100	This goes into the `ci_indexes` table (extra index for the `cube_instance`).

H.3 Oracle Web Services Addressing Properties

Table H–2 lists all the predetermined properties of a normalized message for Web Services Addressing (WS-Addressing).

Table H–2 Properties for Oracle Web Services Addressing

Property Name	Propagatable (Yes/No)	Direction (Inbound /Outbound)	Data Type	Range of Valid Values	Description
`wsa.messageId`	No	Both	String	URI format	This property specifies the identifier for the message and the endpoint to which replies to this message should be sent as an endpoint reference.
`wsa.relatesTo`	No	Both	String	URI format	This optional (repeating) element information item contributes one abstract relationship property value, in the form of an `(IRI, IRI)` pair. The content of this element (of type `xs:anyURI`) conveys the message ID of the related message.
`wsa.replyToAddress`	No	Both	String	URI format	Is a contract between two components communicating asynchronously.

Table H–2 (Cont.) Properties for Oracle Web Services Addressing

Property Name	Propagatable (Yes/No)	Direction (Inbound /Outbound)	Data Type	Range of Valid Values	Description
`wsa.replyToPortType`	No	Both	QName	Any QName	This value is passed to the web service to configure the `portType` on the service's callback. It is translated to the WS-Addressing callback endpoint reference's `PortType` element.
`wsa.replyToService`	No	Both	QName	Any QName	This value is passed to the web service to configure service on the service's callback. It is translated to the WS-Addressing callback endpoint reference's `ServiceName` element.
`wsa.action`	No	Both	String	URI format	This required element (whose content is of type `xs:anyURI`) conveys the value of the action property.
`wsa.to`	No	Both	String	URI format	This optional element (whose content is of type `xs:anyURI`) provides the value for the destination property. If this element is not present, then the value of the (destination) property is `http://www.w3.org/2005/08/addressing/anonymous`.

H.4 Manipulating Normalized Message Properties with bpelx Extensions

Oracle BPEL Process Manager uses `bpelx` extensions to manipulate normalized message properties in message exchange operations. The syntax is different based on whether your BPEL project supports BPEL version 1.1 or 2.0.

H.4.1 BPEL 1.1 bpelx Extensions Syntax

Example H–1 shows `bpelx` extensions syntax in BPEL 1.1.

Example H–1 bpelx Extensions Syntax in Normalized Message Headers in BPEL 1.1

```
<invoke ...>
  <bpelx:inputProperty name="NCName" expression="string" variable="NCName"
 part="NCName" query="string"/>*
  <bpelx:outputProperty name="NCName" expression="string" variable="NCName"
 part="NCName" query="string"/>*
```

```
</invoke>

<receive ...>
  <bpelx:property name="NCName" expression="string" variable="NCName"
 part="NCName" query="string"/>*
</receive>

<onMessage...>
  <bpelx:property name="NCName" expression="string" variable="NCName"
 part="NCName" query="string"/>*
</onMessage>

<reply ...>
  <bpelx:property name="NCName" expression="string" variable="NCName"
 part="NCName" query="string"/>*
</reply>
```

H.4.2 BPEL 2.0 bpelx Extensions Syntax

Example H–2 shows `bpelx` extensions syntax in BPEL 2.0.

Example H–2 bpelx Extensions Syntax in Normalized Message Headers in BPEL 2.0

```
<invoke ...>
  <bpelx:fromProperties>?
    <bpelx:fromProperty name="NCName" .../>+
  </bpelx:fromProperties>
  <bpelx:toProperties>?
    <bpelx:toProperty name="NCName" .../>+
  </bpelx:toProperties>
</invoke>

<receive ...>
  <bpelx:fromProperties>?
    <bpelx:fromProperty name="NCName" .../>+
  </bpelx:toProperties>
</receive>

<onEvent ...>
  <bpelx:fromProperties>?
    <bpelx:fromProperty name="NCName" .../>+
  </bpelx:fromProperties>
</onEvent>

<reply...>
  <bpelx:toProperties>?
    <bpelx:toProperty name="NCName" .../>+
  </bpelx:toProperties>
</reply>

<reply ...>
  <bpelx:toProperties>
    <bpelx:toProperty name="NCName" .../>
  </bpelx:toProperties>
</reply>
```

Note the following details:

- The `toProperty` is a `from-spec`. This copies a value from the `from-spec` to the property of the given name.

- The `fromProperty` is a `to-spec`. This copies a value from the property to the `to-spec`.

Interfaces Implemented By Rules Dictionary Editor Task Flow

Oracle Business Rules Dictionary Editor Task Flow implements two interfaces when creating an ADF-based Web application. The interfaces are defined in the `soaComposerTemplates.jar` file.

This appendix includes the following sections:

I.1 The MetadataDetails Interface

The `MetadataDetails` interface is a part of the `oracle.integration.console.metadata.model.share` package and is defined in the `soaComposerTemplates.jar` file.

The `MetadataDetails` interface defines three methods, as shown in Example I–1:

Example I–1 MetadataDetails Interface

```
public interface MetadataDetails {
    /**
     * Retrieve the details of the metadata document
     * @return document in string format.
     */
    String getDocument();

    /**
     * Get related document.
     */
    String getRelatedDocument(final RelatedMetadataPath relatedPath);

    /**
     * Update the metadata document.
     * @param doc represents the updated document.
     */
    void setDocument(String doc) throws Exception;
}
```

I.1.1 The getDocument Method

This method is used to retrieve the rules file in a string format. For doing this action, you must connect to the Oracle Metadata Repository (MDS) or a file system, and return the rules file in a string format.

Example I–2 shows how to get the file from a local file system:

Example I–2 getDocument Method

```
private static final String RULES_FILE1 =
"file:///C:/scratch/<username>/system/mywork/linkedD/AutoAppProj/oracle/rules/cred
it/CreditRatingRules.rules";

   public String getDocument() {
        URL url = null;
        try {
            url = new URL(RULES_FILE1);
            return readFile(url);
        } catch (IOException e) {
            System.err.println(e);
        }
        return "";
    }

    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();
    }
```

I.1.2 The getRelatedDocument Method

This method is required when you work with linked dictionaries. You must connect to MDS, find the related dictionary file, and then return it in a string format. Example I–3 shows how to find the path of the linked dictionaries that are stored within the `../oracle/rules` directory in a local file system:

Example I–3 getRelatedDocument Method

```
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 "";
    }
```

I.1.3 The setDocument Method

This method is used to store the rules file. It returns a String `doc` value, which is the name of the updated dictionary based on user edits performed by using Rules Dictionary Editor Task Flow. You must store the rules file in MDS or a file system. Example I–4 shows how to save the document in the local file system:

Example I–4 setDocument Method

```
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);
                }
            }
        }
    }
```

I.2 The NLSPreferences Interface

The `NLSPrefrences` interface defines four methods as shown in Example I–5:

Example I–5 NLSPreferences Interface

```
public interface NLSPreferences
{
   /**
    * Returns the locale to be used.
    **/
    Locale getLocale();

    /**
     * Return the timezone to be used.
     **/
    TimeZone getTimeZone();

    /**
     * Return the dateformat to be used.
     */
    String getDateFormat();

    /**
     * Return the time format to be used.
     */
    String getTimeFormat();
}
```

Example I–6 is a sample implementation of the `NLSPreferences` interface:

Example I–6 Sample Implementation of the NLSPreferences Interface

```
public class MyNLSPreferences implements NLSPreferences {
        private static final String DATE_STYLE = "yyyy-MM-dd";
        private static final String TIME_STYLE = "HH-mm-ss";

        public Locale getLocale() {
            return Locale.FRENCH;
        }

        public TimeZone getTimeZone() {
            return TimeZone.getTimeZone("America/Los_Angeles");
        }

        public String getDateFormat() {
            return DATE_STYLE;
        }
```

```
    public String getTimeFormat() {
        return TIME_STYLE;
    }
}
```

J

Oracle User Messaging Service Applications

This appendix describes how to create your own Oracle User Messaging Service applications using the procedures and code provided.

This appendix includes the following sections:

- Section J.1, "Send Message to User Specified Channel"
- Section J.2, "Send Email with Attachments"

> **Note:** To learn more about the code samples for Oracle User Messaging Service, or to run the samples yourself, refer to the Oracle Technology Network code sample page at the following URL:
> `https://soasamples.samplecode.oracle.com/`
>
> Once you have navigated to this page, you can find code samples for Oracle User Messaging Service by entering the search term "UMS" and clicking **Search**.

J.1 Send Message to User Specified Channel

This chapter describes how to build and run the Send Message to User Specified Channel application provided with Oracle User Messaging Service.

> **Note:** To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This section contains the following subsections:

- Section J.1.1, "Overview"
- Section J.1.2, "Installing and Configuring SOA and User Messaging Service"
- Section J.1.3, "Building the Sample"
- Section J.1.4, "Creating a New Application Server Connection"
- Section J.1.5, "Deploying the Project"
- Section J.1.6, "Configuring User Messaging Preferences"
- Section J.1.7, "Testing the Sample"

J.1.1 Overview

The "Send Message to User Specified Channel" application demonstrates a BPEL process that allows a message to be sent to a user through a messaging channel specified in User Messaging Preferences. After you have configured a device and messaging channel addresses for each supported channel and the default device, Oracle User Messaging Service routes the message to the user based on the preferred channel setting that you configured.

J.1.1.1 Provided Files

The following files are included in the application:

- SendMessage.pdf – this document.
- Project – the directory containing Oracle JDeveloper project files.
- Readme.txt.
- Release notes

J.1.2 Installing and Configuring SOA and User Messaging Service

The installation of SOA and User Messaging Service has already been performed on your hosted instance, and the sample users have already been seeded. Perform the following steps to enable notifications in soa-infra, if not already done:

1. Using Enterprise Manager, go to the **SOA Infrastructure** menu, and select **SOA Administration** > **Workflow Notification Properties**, and set **Notification Mode** to **ALL**.
2. Configure the User Messaging drivers if required as described in "Configuring Drivers" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.
3. Set the email address for user `weblogic` by using the JXplorer LDAP browser. Refer to "Updating Addresses in Your LDAP User Profile".
4. Restart the server.

J.1.2.1 Updating Addresses in Your LDAP User Profile

Perform the following steps to set the email address for user `weblogic` by using the JXplorer LDAP browser:

J.1.2.1.1 Installing Download and install JXplorer from `http://www.jxplorer.org`.

J.1.2.1.2 Connecting 1.Set the embedded LDAP server admin password as follows:

- Login to the Oracle WebLogic Server Administration Console.
- Click the domain name link > **Security** > **Embedded LDAP.**
- Enter a new **Credential** and **Confirm Credential** (for example, `weblogic`).
- Click **Save**.

2. Connect from JXplorer by specifying the fields in Table J–1:

Table J–1 JXplorer Connection Fields

Field	Value
Host	Oracle WebLogic Administration Server hostname
Port	Oracle WebLogic Administration Server port
Protocol	LDAP v3
Security Level	User + Password
User DN	cn=Admin
Password	password

J.1.2.1.3 Setting User Messaging Device Addresses in LDAP The following example uses the user `weblogic`. You may create and use additional users.

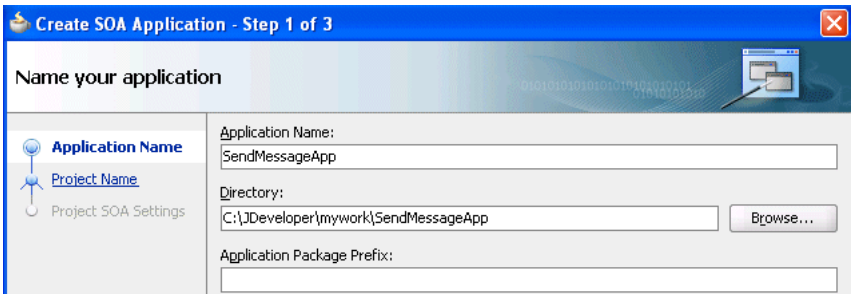
1. Expand the LDAP tree as follows: **domain** > **myrealm** > **people** > **weblogic**.
2. Click the user entry.
3. Select the HTML view tab on the right.
4. Enter the necessary Email Address and Mobile Phone Number.
5. Click **Submit**.

J.1.3 Building the Sample

Performing the following procedure of building the sample from scratch enables you to learn how to add messaging to your SOA Composite Applications, and use User Messaging Preferences.

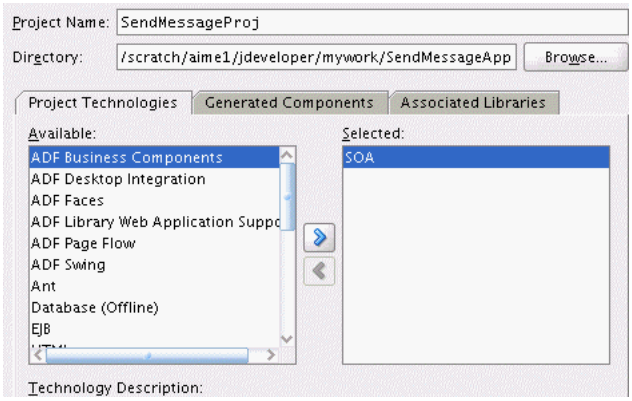
1. Open Oracle JDeveloper 11g.
2. Create a new application by selecting **File, New**, **General**, **Applications**, and **SOA Application**. Click **OK**.
3. Enter the *Application Name* and click **Next** (Figure J–1).

Figure J–1 Creating a New Application and Project (1 of 3)



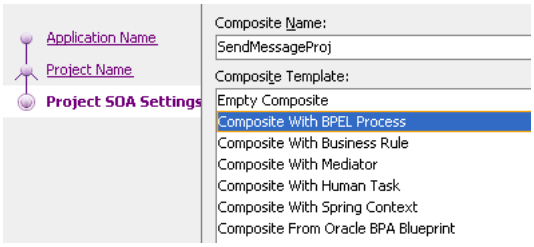
4. Enter the name for the project and click **Next** (Figure J–2).

Figure J–2 Creating a New Application and Project (2 of 3)



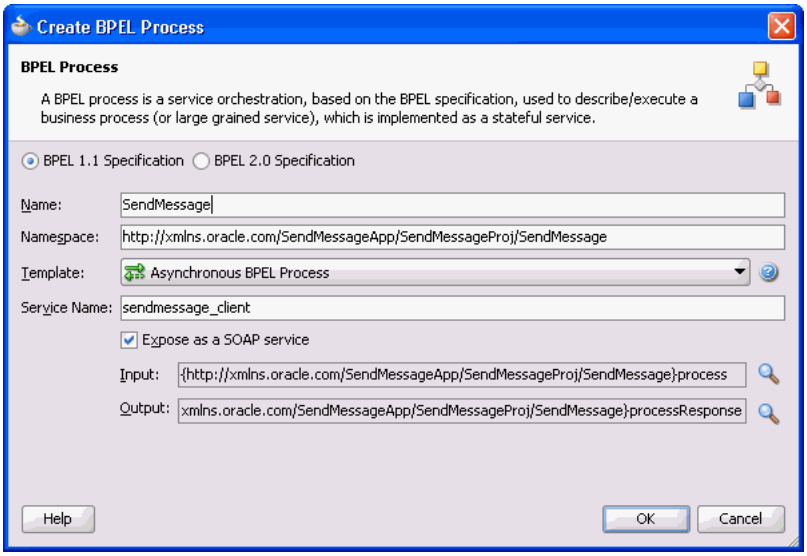
5. Select the **Composite With BPEL** composite template (Figure J–3). Click **Finish**.

Figure J–3 Creating a New Application and Project (3 of 3)



6. In the Create BPEL Process dialog, enter the BPEL process name as `SendMessage` (Figure J–4). Click **OK**.

Figure J–4 Creating the BPEL Process



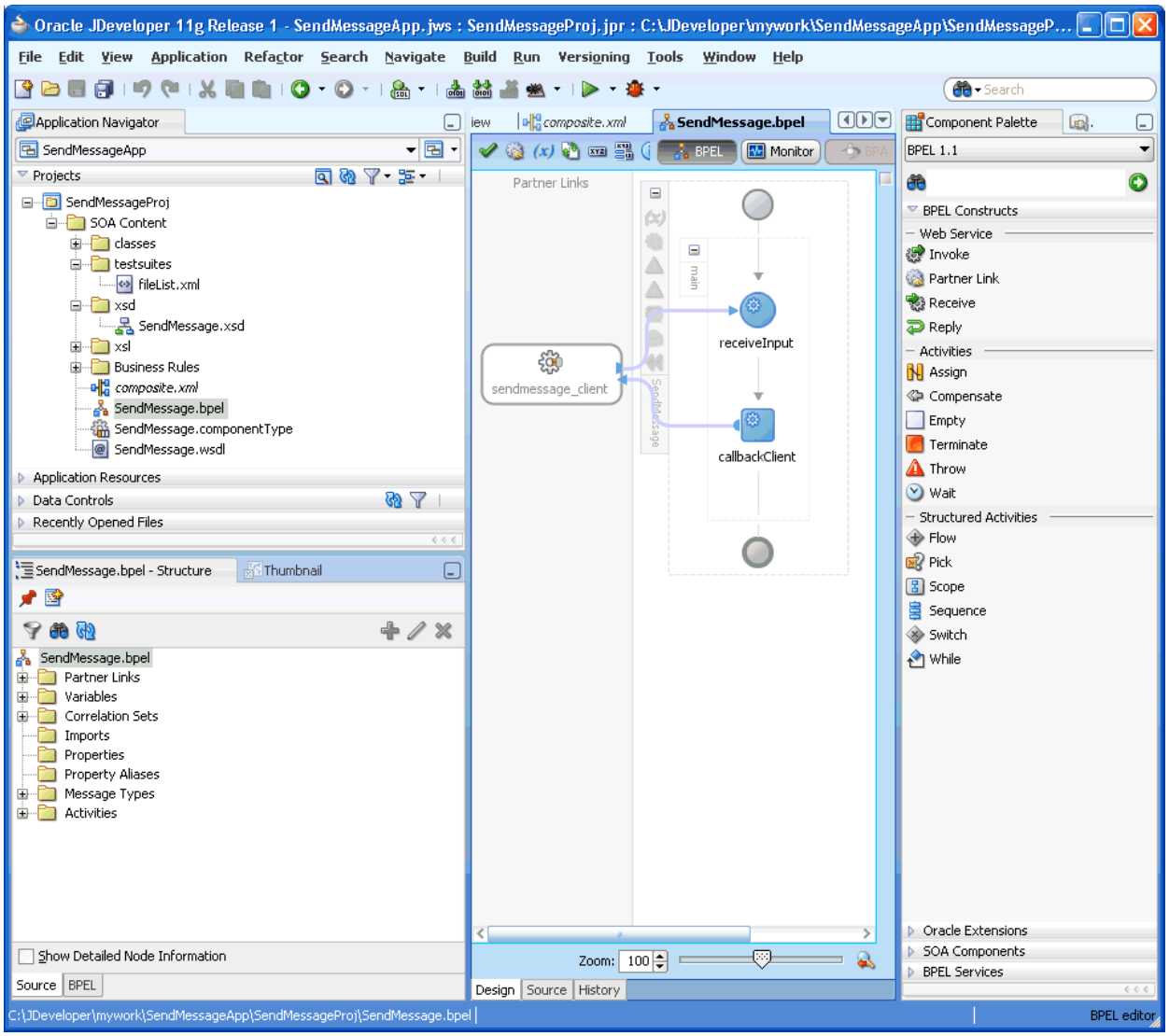
7. Verify that **Expose as a SOAP service** is checked. Click **OK**.
8. You have now created an empty and default BPEL application (Figure J–5).

In the Oracle JDeveloper main window you can view the following components of the application under the **Composite.xml** tab.

- The left box is the definition of a web service client that is used to initiate an application.
- The middle box is a BPEL process that creates and formats the message and calls the messaging service.

Note: You later create the messaging service resource that is used to send the message when you create the User Notification BPEL process (steps 13 - 19).

Figure J–5 Empty and Default BPEL Application



9. Expand the **xsd** folder in the Application Navigator and open **SendMessage.xsd** by double-clicking it.
10. Click the **Source** tab (Figure J–6).
11. Perform the following modifications to the inputs of this BPEL application:

In the generated file, **SendMessage.xsd**, in the **xsd** folder in the Application Navigator under projects, the following element definition is created by default:

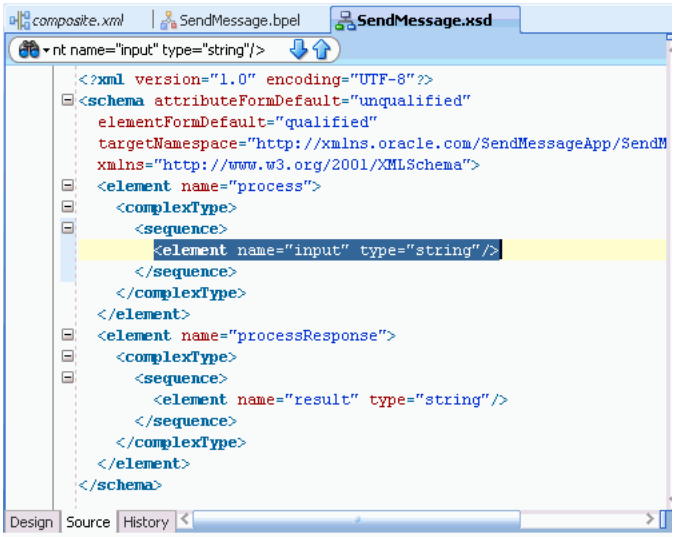
```
<element name="input" type="string"/>
```

This XSD element defines the input for the BPEL process.

Select the **Source** tab (Figure J–6), and replace the line above with the following three lines:

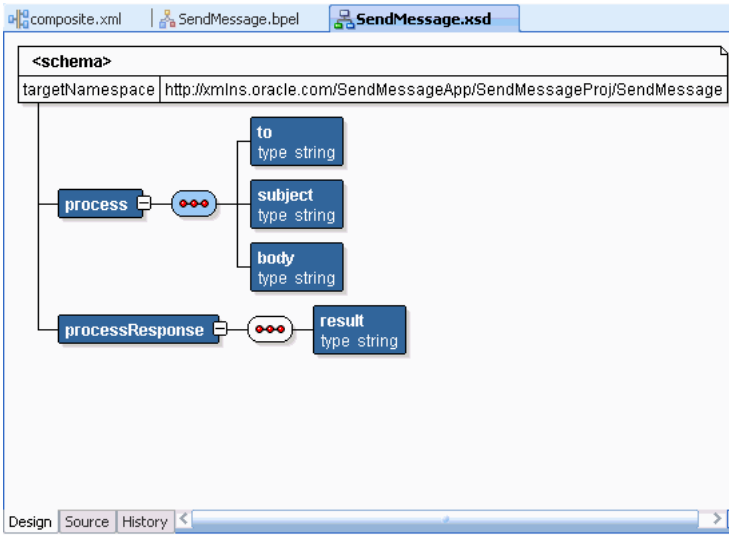
```
<element name="to" type="string"/>
<element name="subject" type="string"/>
<element name="body" type="string"/>
```

Figure J–6 Modifying the Inputs in the SendMessage.xsd File



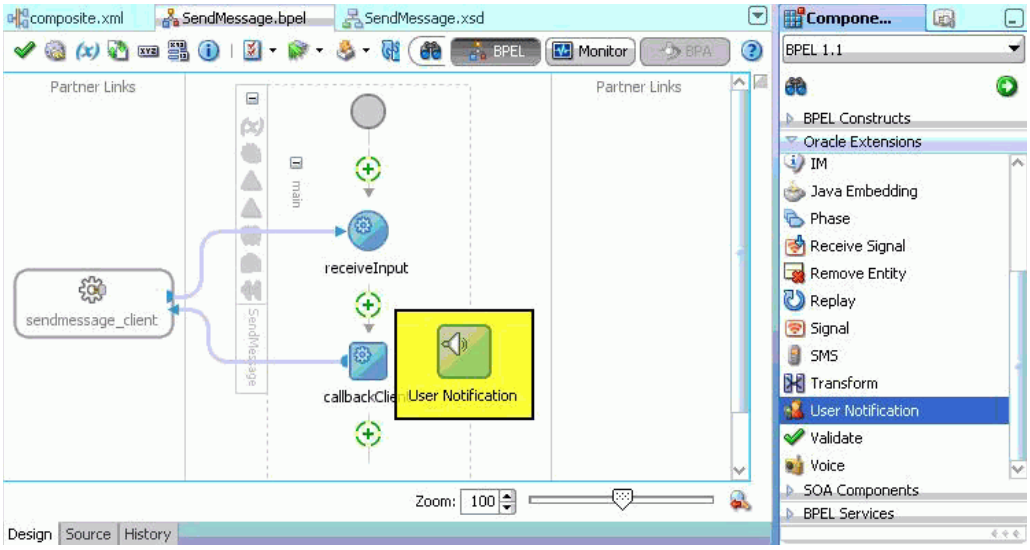
12. From the **File** menu, select **Save All**.
13. View the expanded process element (Figure J–7).

Figure J–7 Viewing the Expanded Process Element



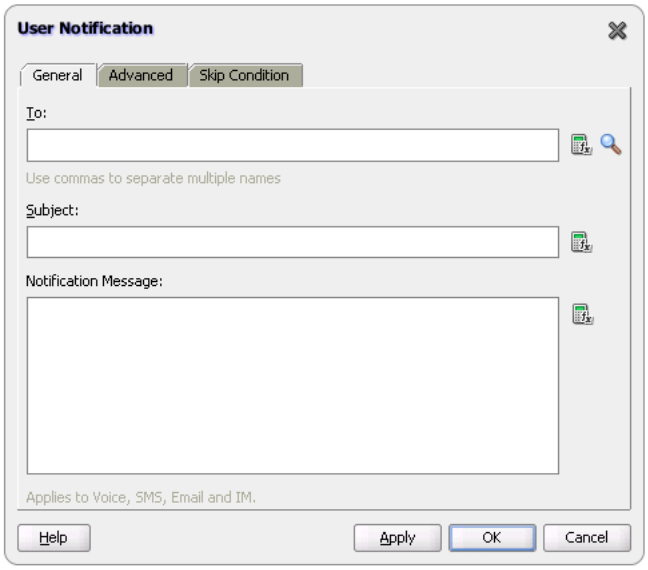
14. To enable messaging in this process, drag and drop **User Notification** from **Oracle Extensions** located in the Component Palette between the **receiveInput** and **callbackClient** activities (Figure J–8).

Figure J–8 Dragging and Dropping User Notification Icon from the Component Palette



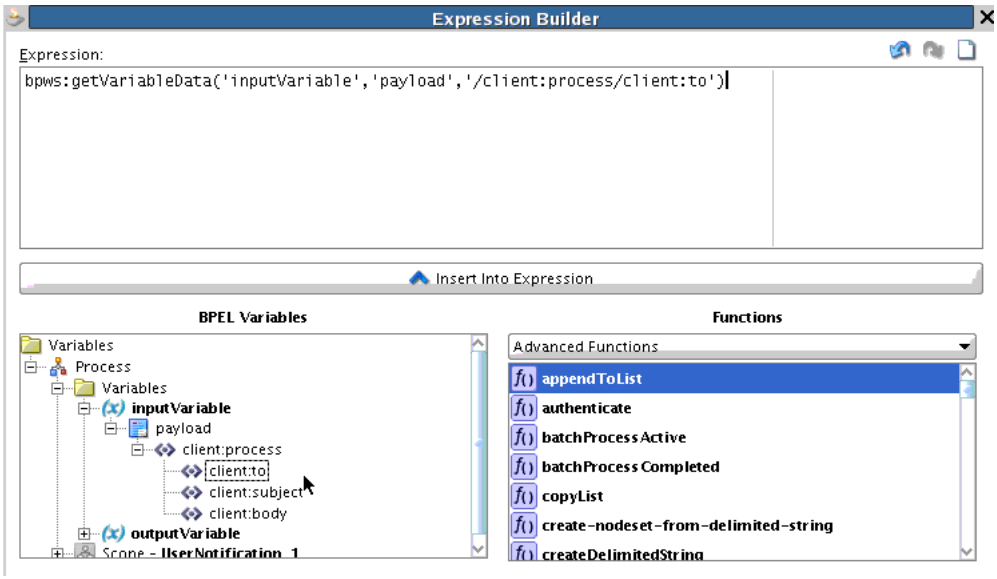
The User Notification activity appears (Figure J–9).

Figure J–9 User Notification Activity Before Configuring the Inputs



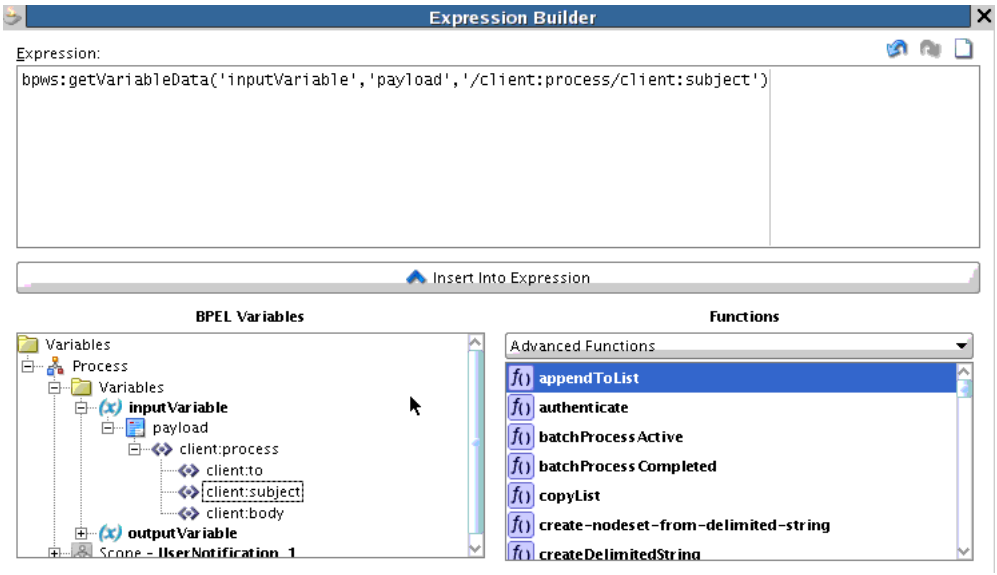
15. Click the XPath Expression Builder icon to the right of the **To:** input box.
16. Modify the expression for the **To** recipient, as follows:
    - In the BPEL Variables pane, select **Variables**, **inputVariable**, **Payload**, **clientprocess**, and **client:to** (Figure J–10).
    - Click **Insert Into Expression**.
    - Click **OK**.

Figure J–10 Defining the Recipient ("to") Expression



17. Click the XPath Expression Builder icon to the right of the **subject:** input box.
18. Modify the expression for the subject as follows:
    - In the BPEL Variables pane, select **Variables**, **InputVariable**, **Payload**, **clientprocess**, and **client:subject** (Figure J–11).
    - Click **Insert Into Expression**.
    - Click **OK**.

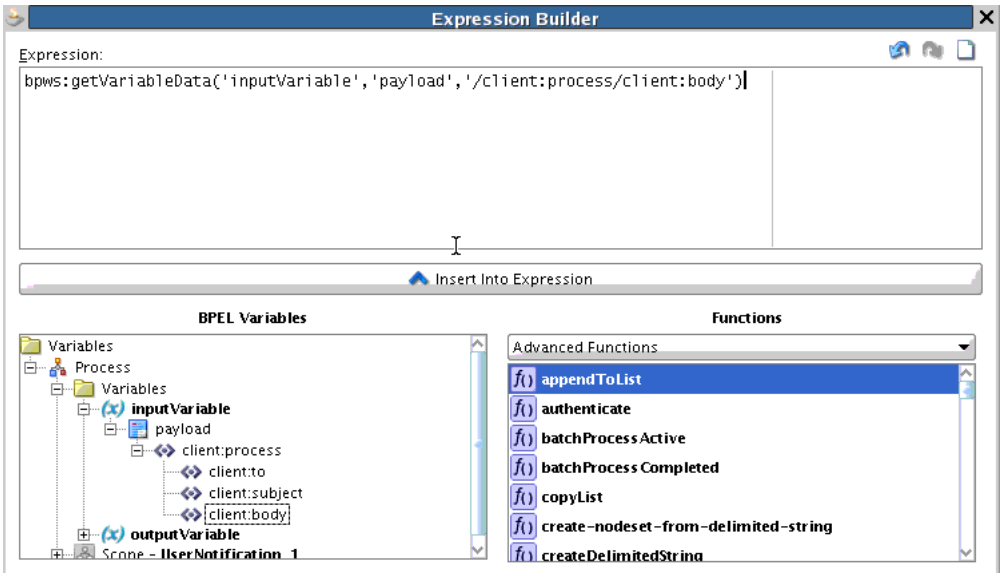
Figure J–11 Defining the Subject Expression



19. Click the XPath Expression Builder icon to the right of the **Notification Message:** input box.
20. Modify the expression for the notification message as follows:

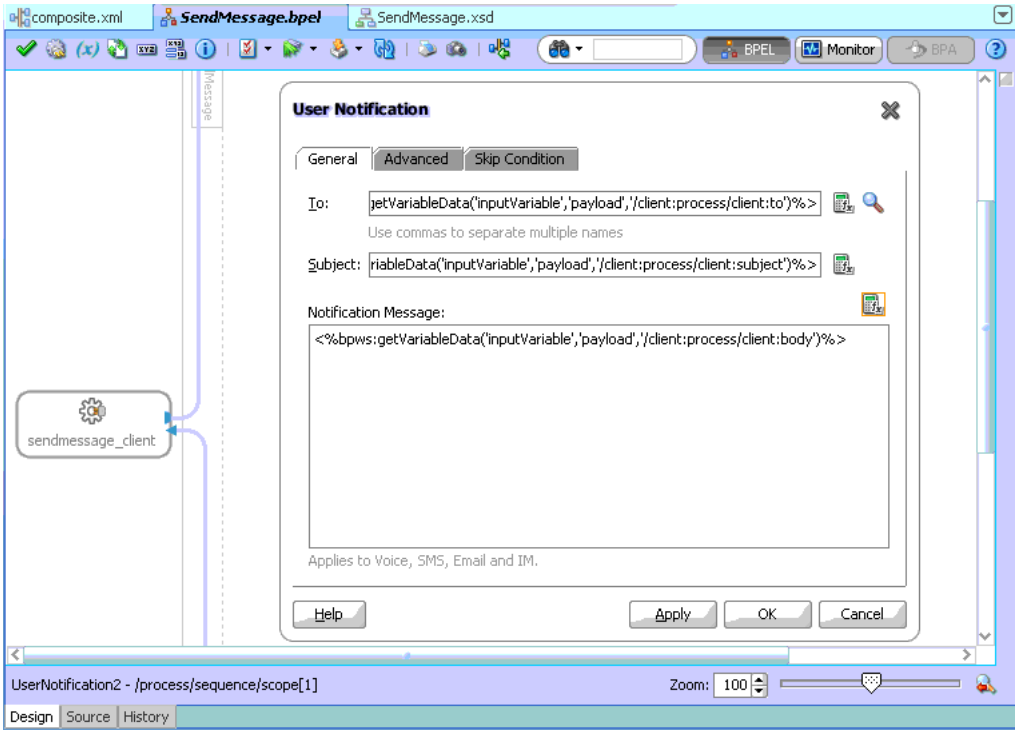
- In the BPEL Variables pane, select **Variables**, **InputVariable**, **Payload**, **clientprocess**, and **client:body** (Figure J–12).
- Click **Insert Into Expression**.

Figure J–12 Defining the Body Expression



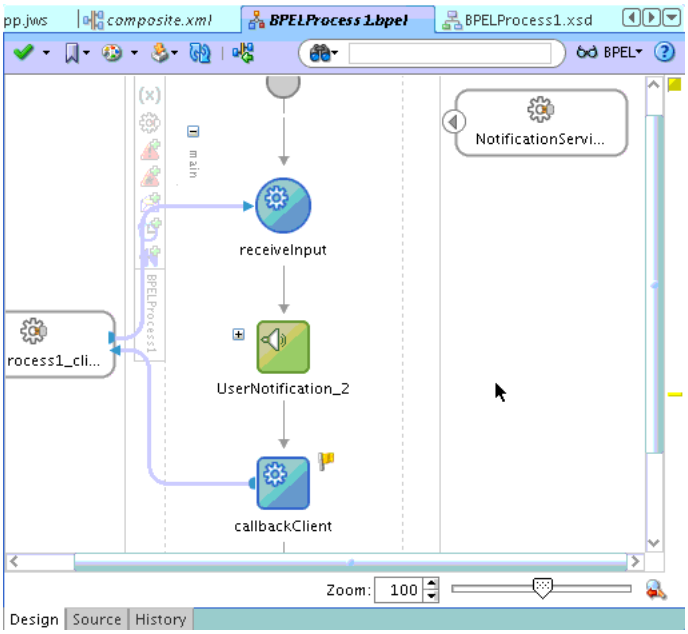
- Click **OK**.
- Click **Apply and then OK** to apply the changes (Figure J–13).

Figure J–13 Confirming the Changes to the Inputs



The changes to the inputs are saved and the configuration of the User Notification Activity is complete. You can now see the **User Notification** activity in the BPEL application (Figure J–14). The SOA Composite is complete.

Figure J–14 User Notification Activity After Configuration of Inputs

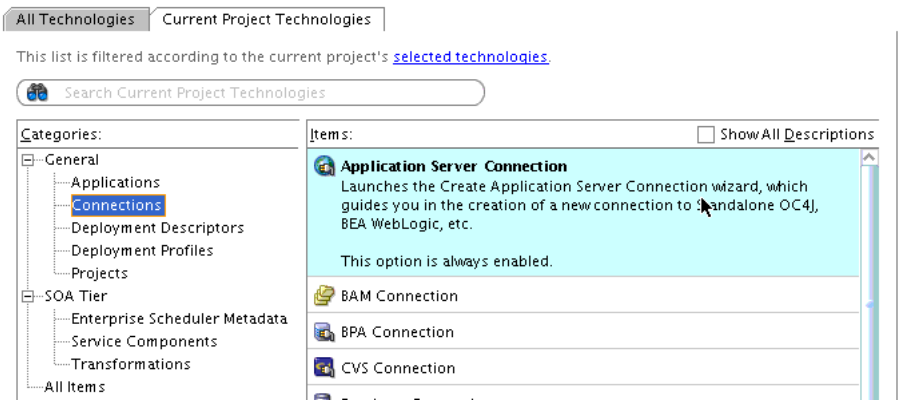


J.1.4 Creating a New Application Server Connection

Perform the following steps to create an Application Server Connection.

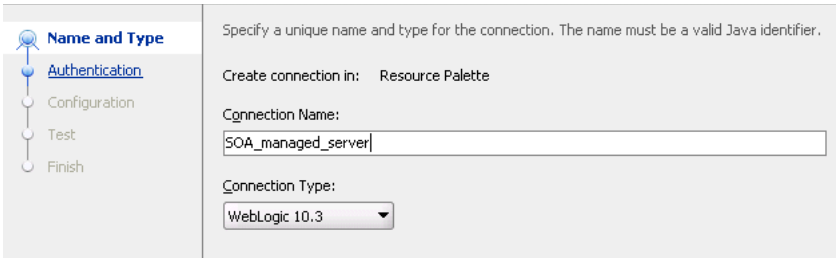
1. Create a new Application Server Connection by right-clicking the project and selecting **New**, **Connections**, and **Application Server Connection** (Figure J–15).

Figure J–15 New Application Server Connection



2. Name the connection `SOA_server` and click **Next** (Figure J–16).
3. Select **WebLogic 10.3** as the **Connection Type**.

Figure J–16 New Application Server Connection



4. Enter the authentication information. The typical value for username is `weblogic`.
5. In the Connection dialog, enter the hostname, port and SSL port for the SOA admin server, and enter the name of the domain for the Oracle WebLogic Server Domain.
6. Click **Next**.
7. In the Test dialog, click **Test Connection**.
8. Verify that the message `Success!` appears.

   The application server connection has been created.

J.1.5 Deploying the Project

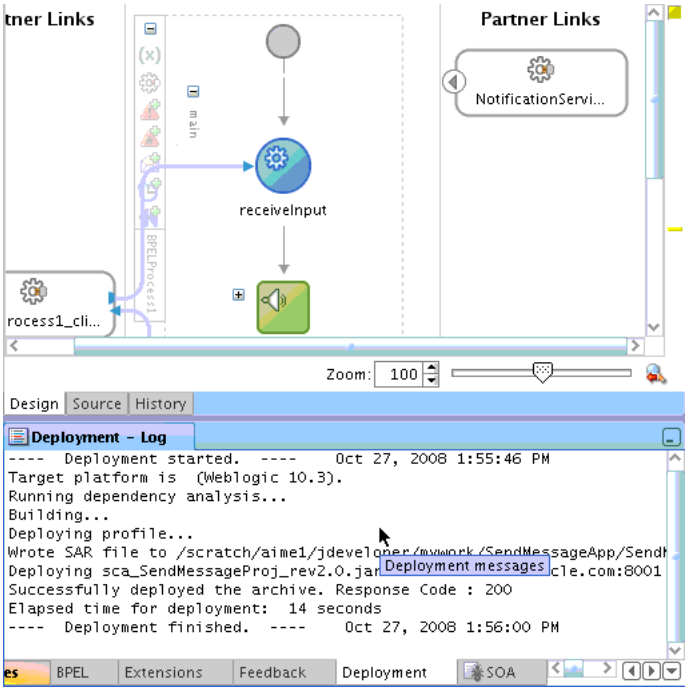
Perform the following steps to deploy the project:

1. In the Application Navigator, right-click the SOA project.
2. Select **Deploy** > **project_name**.

   The value for project_name is the SOA project name.

   The Deployment Action page of the Deploy Project_Name wizard appears.
3. Select **Deploy to Application Server**.
4. Click **Next**.
5. Select an existing connection to an application server from the list.
6. Click **Finish**.
7. Verify that the message `Build Successful` appears in the log.
8. Enter the default revision and click **OK**.
9. Verify that the message `Deployment Finished` appears in the deployment log (Figure J–17).

Figure J–17 Verifying that the Deployment is Successful



You have successfully deployed the application.

Before you can run the sample you must configure any additional drivers in Oracle User Messaging Service and configure a default device for the user receiving the message in User Messaging Preferences, as described in the following sections.

Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

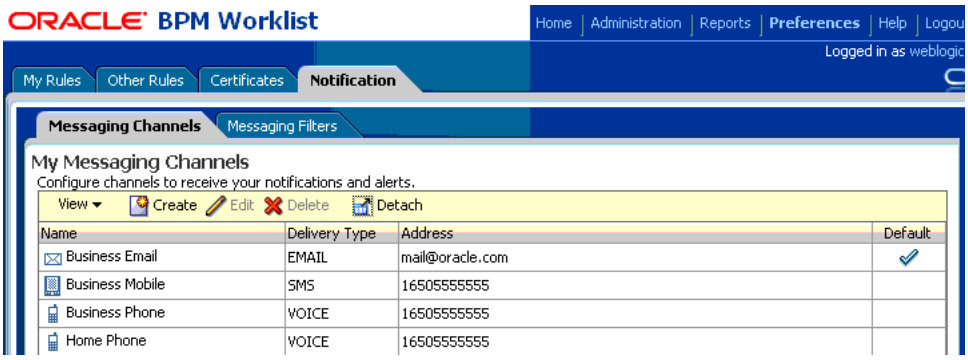
J.1.6 Configuring User Messaging Preferences

For users to receive the notifications, they must register the devices that they use to access messages through User Messaging Preferences. Perform the following steps:

1. Log in to the User Messaging Preferences application at one of the following URLs:
   - Directly at `http://server:port/sdpmessaging/userprefs-ui`
   - Through the Worklist application's Preferences > Notification tab at: `http://server:port/integration/worklistapp`

   The User Messaging Preferences application appears.
2. Click the **Messaging Channels** tab (Figure J–18).

Figure J–18 Messaging Channels Tab



You are prompted for login credentials.

3. In the **Messaging Channels** tab, select a channel.
4. Set a channel as the default by expanding the device folder, and then clicking **Set as Default** adjacent to the selected channel.

   A check mark appears next to the selected channel, designating it as the default means of receiving notifications. All messages sent to that user are sent to that channel.

J.1.7 Testing the Sample

The following steps describe how to perform a test message transmission through Enterprise Manager.

Perform the following steps to run and test the sample:

1. Open a web browser window and login to Enterprise Manager for the SOA domain. For example, `http://host:port/em`.
2. In Oracle Enterprise Manager, expand the SOA folder in the navigation tree, and click the deployed **SendMessageProj** composite application. Click the **Test** button to launch the test client page.
3. In the **Input Arguments** section provide the input values for invoking **SendMessageProj**.

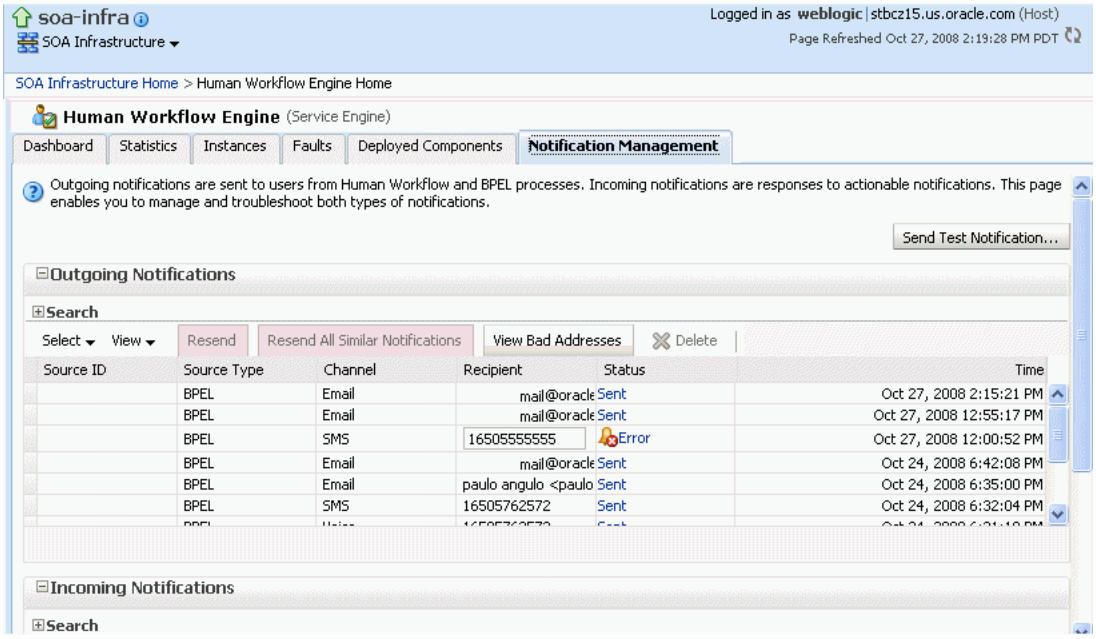
   Enter the following values:

   - **to:** weblogic (the user)
   - **subject:** notification test (the subject)
   - **body:** the message content
4. Click **Test Web Service**.

J.1.7.1 Verifying the Execution of Sending the Email

Log in to the Human Workflow Engine. Verify the outgoing notifications and their statuses from the Notification Manager tab. (Figure J–19).

Figure J–19 Viewing Outgoing Notifications



J.2 Send Email with Attachments

This section describes how to build and run the Send Email with Attachments application provided with Oracle User Messaging Service.

> **Note:** To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This section contains the following subsections:

- Section J.2.1, "Overview"
- Section J.2.2, "Installing and Configuring SOA and User Messaging Service"
- Section J.2.3, "Running the Pre-Built Sample"
- Section J.2.4, "Testing the Sample"
- Section J.2.5, "Building the Sample"
- Section J.2.6, "Creating a New Application Server Connection"

J.2.1 Overview

The "Send Email With Attachment" application demonstrates a BPEL process that sends an email with an attached file.

A BPEL process looks up a user's email address from the identity store, reads a file from the file system, creates email content and then sends an email to the user.Section J.2.5, "Building the Sample" shows you how to add an email with attachments to your SOA composite application, allowing your applications to be enabled with messaging.If you want to model the application from scratch, go to the section titled Building the Sample. Or, you can directly use the pre-built project provided with this tutorial.

Before you run the pre-built sample or build the application from scratch, you must install and configure the server as described in Section J.2.2, "Installing and Configuring SOA and User Messaging Service". By default, soa-infra does not send out notifications. The following steps describe installing and configuring the email drivers needed to communicate with the email server.

J.2.1.1 Provided Files

The following files are included in the sample application:

- ns_sendemail.pdf – this document.
- Project – the directory containing Oracle JDeveloper project files.
- Readme.txt.
- Release notes

J.2.2 Installing and Configuring SOA and User Messaging Service

The installation of SOA and User Messaging Service has already been performed on your hosted instance, and the sample user, `weblogic`, has already been created. Perform the following steps to enable notifications in soa-infra, if not already done:

1. Using Enterprise Manager, go to the **SOA Infrastructure** menu, and select **SOA Administration** > **Workflow Notification Properties**, and set **Notification Mode** to **ALL**.
2. Configure the User Messaging drivers if required as described in "Configuring Drivers" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.
3. Set the email address for user `weblogic` by using the JXplorer LDAP browser. Refer to "Updating Addresses in Your LDAP User Profile".
4. Restart the server.

J.2.2.1 Updating Addresses in Your LDAP User Profile

Perform the following steps to set the email address for user `weblogic` by using the JXplorer LDAP browser:

J.2.2.1.1 Installing Download and install JXplorer from http://www.jxplorer.org.

J.2.2.1.2 Connecting 1.Set the embedded LDAP server admin password as follows:

- Login to the Oracle WebLogic Server Administration Console.
- Click the domain name link > **Security** > **Embedded LDAP**.
- Enter a new Credential and Confirm Credential (for example, `weblogic`).
- Click **Save**.

2. Connect from JXplorer by specifying the fields in Table J–2:

Table J–2 JXplorer Connection Fields

Field	Value
Host	Oracle WebLogic Administration Server hostname
Port	Oracle WebLogic Administration Server port

Table J–2 (Cont.) JXplorer Connection Fields

Field	Value
Protocol	LDAP v3
Security Level	User + Password
User DN	cn=Admin
Password	password

J.2.2.1.3 Setting User Messaging Device Addresses in LDAP The following example uses the user `weblogic`. You may create and use additional users.

1. Expand the LDAP tree as follows: **domain** > **myrealm** > **people** > **weblogic**.
2. Click the user entry.
3. Select the HTML view tab on the right.
4. Enter the necessary Email Address and Mobile Phone Number.
5. Click **Submit**.

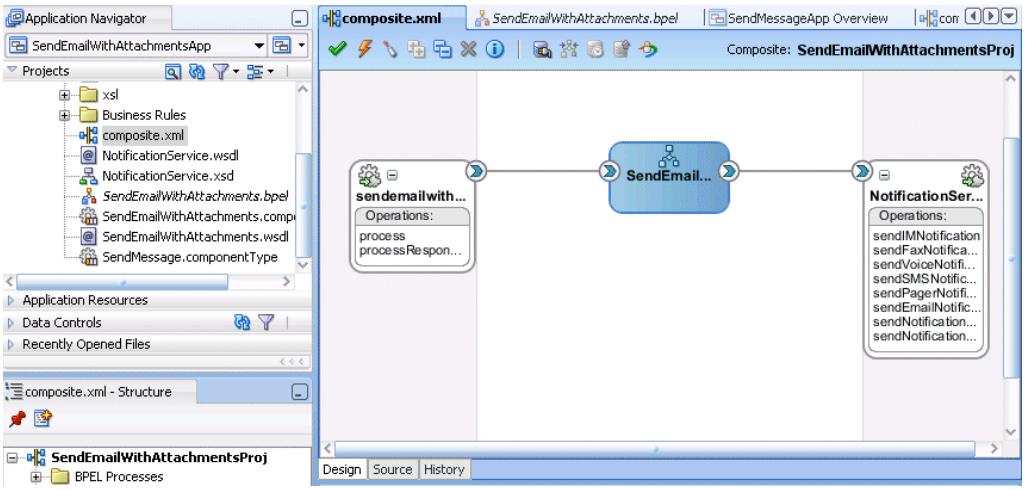
J.2.3 Running the Pre-Built Sample

Perform the following steps to run and deploy the prebuilt sample application:

1. Open **SendEmailWithAttachmentsApp.jws** (contained in the .zip file) in Oracle JDeveloper.

   In the Oracle JDeveloper main window you can view the following components of the sample application under the **Composite.xml** tab.

Figure J–20 Oracle JDeveloper Main Window



- The left box is the definition of a web service client that is used to initiate an application.
- The middle box is a BPEL process that creates and formats the message and calls the messaging service.
- The right box is the messaging service resource that is used to send the message.

2. Create an Application Server Connection by right-clicking the project in the navigation pane and selecting New. Follow the instructions in Section J.2.6, "Creating a New Application Server Connection."
3. Deploy the project as follows:
    1. In the Application Navigator, right-click the SOA project.
    2. Select **Deploy** > **project_name**.

       The value for project_name is the SOA project name.

       The Deployment Action page of the Deploy Project_Name wizard appears.
    3. Select **Deploy to Application Server**.
    4. Click **Next**.
    5. Select an existing connection to an application server from the list.
    6. Click **Finish**.
4. Verify that the message `Build Successful` appears in the log.
5. Enter the default revision and click **OK**.
6. Verify that the message `Deployment Finished` appears in the deployment log.

   You have successfully deployed the application.

   Before you can run the sample you must configure any additional drivers in Oracle User Messaging Service and configure a default device for the user receiving the message in User Messaging Preferences, as described in the following sections.

   Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite* for more information.

J.2.4 Testing the Sample

The following steps describe how to perform a test message transmission through Enterprise Manager.

Perform the following steps to run and test the sample:

1. Open a web browser window and login to Enterprise Manager for the SOA domain. For example, `http://host:port/em`.
2. In Enterprise Manager, expand the SOA folder in the navigation tree, and click the deployed **SendEmailWithAttachmentsProj** composite application. Click the **Test** button to launch the test client page.
3. In the **Input Arguments** section provide the input values for invoking **SendEmailWithAttachmentsProj**.

   Enter the following values:

   - **to:** weblogic (the user)
   - **subject:** notification test (the subject)
   - **body:** the message content
   - **attachmentName**: the name of the file being attached, including extension.
   - **attachmentMimeType**: for example, image/gif.

To send binary files such as PDF, DOC, GIF, or JPEG files, the following values can be used for the attachmentMimeType entry:

- file-name.doc – attachmentMimeType: application/msword
- file-name.pdf – attachmentMimeType: application/pdf
- file-name.jpg – attachmentMimeType: image/jpeg
- file-name.gif – attachmentMimeType: image/gif

To send text files such as HTML, XML, or plain text files, the following values can be used for the attachmentMimeType entry:

- file-name.txt – attachmentMimeType: text/plain
- file-name.html – attachmentMimeType: text/html

Note: For text files that contain non-ASCII characters that are encoded in UTF-8, the attachmentMimeType must specify the charset attribute, for example, "text/plain;charset=UTF-8". Also, the content itself must be sent using base64 encoding; this procedure is described in "Sending Text Content with base64 Encoding".

- attachmentURI: the URI for the attachment

4. Click **Test Web Service**.

J.2.4.1 Verifying the Execution

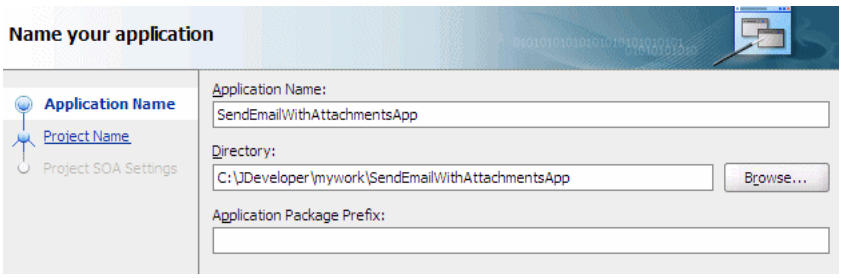
Check the weblogic email account. It should have received an email with attachment.

J.2.5 Building the Sample

Performing the following procedure of building the sample from scratch enables you to learn how to add messaging to your SOA Composite Applications, and use User Messaging Preferences.

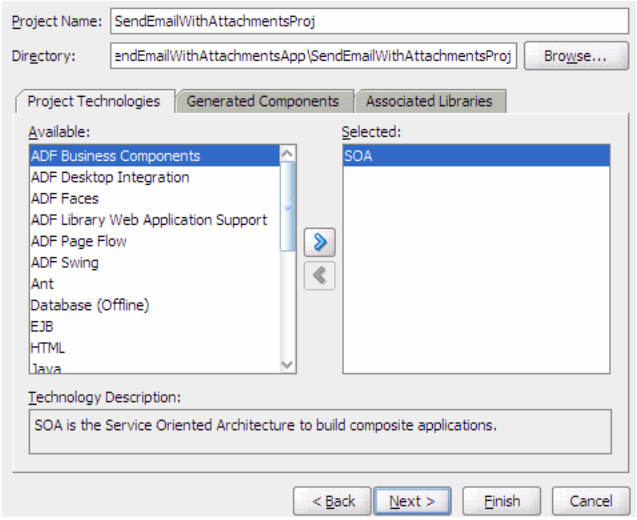
1. Open Oracle JDeveloper 11g.
2. Create a new application by selecting **File, New**, **Applications**, and **SOA Application**. Click **OK**.
3. Enter the *Application Name* and click **Next** (Figure J–21).

Figure J–21 Creating a New Application and Project (1 of 3)



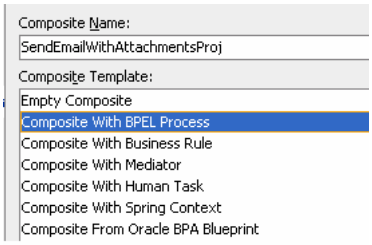
4. Enter the name for the project and click **Next** (Figure J–22).

Figure J–22 Creating a New Application and Project (2 of 3)



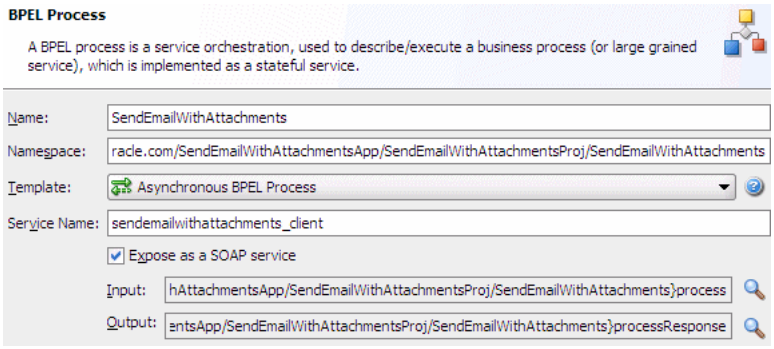
5. Select the **Composite With BPEL Process** composite template (Figure J–23). Click **Finish**.

Figure J–23 Creating a New Application and Project (3 of 3)



6. In the Create BPEL Process dialog, enter the BPEL process name as `SendEmailWithAttachments` (Figure J–24). Click **OK**.

Figure J–24 Creating the BPEL Process



7. Verify that **Expose as a SOAP service** is checked. Click **OK**.
8. You have now created an empty and default BPEL application.

   In the Oracle JDeveloper main window you can view the following components of the sample application under the *Composite.xml* tab.

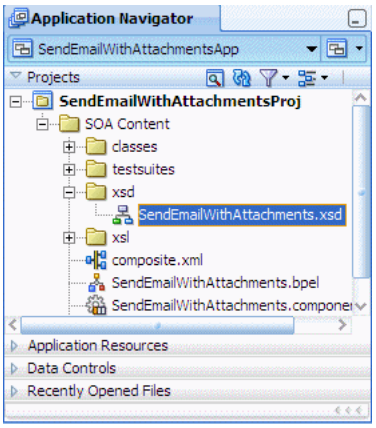
- The left box is the definition of a web service client that is used to initiate an application.
- The middle box is a BPEL process that creates and formats the message and calls the messaging service.

Note: You later create the messaging service resource that is used to send the message when you create the User Notification BPEL process (steps 13-19).

---

9. Expand the **xsd** folder in the Application Navigator and open **SendEmailWithAttachments.xsd** by double-clicking it (Figure J–25).

Figure J–25 Accessing the SendEmailWithAttachments.xsd File



10. Click the **Source** tab (Figure J–25).
11. Perform the following modifications to the inputs of this BPEL application:

    In the generated file, **SendEmailWithAttachments.xsd**, in the xsd folder in the Application Navigator under projects, the following element definition is created by default:

```
<element name="process">
  <complexType>
    <sequence>
      <element name="input" type="string"/>
    </sequence>
  </complexType>
</element>
```

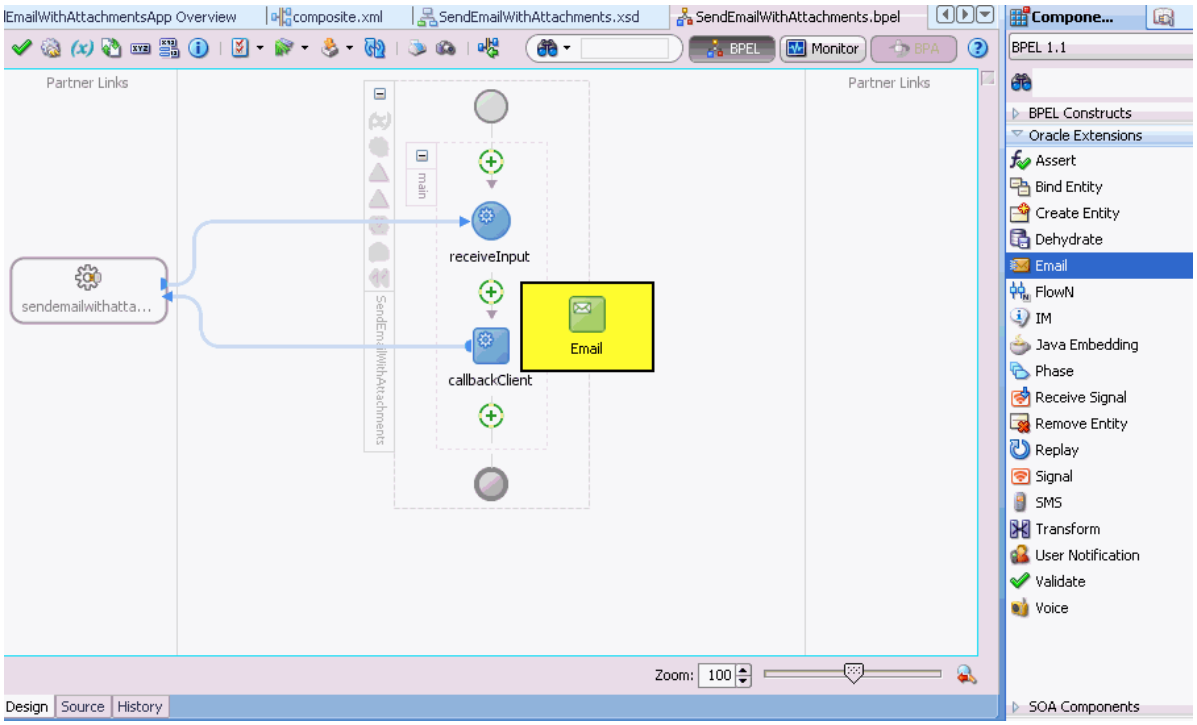
    Select the **Source** tab, and replace the lines above with the following:

```
<element name="process">
<complexType>
     <sequence>
       <element name="to" type="string"/>
       <element name="subject" type="string"/>
       <element name="body" type="string"/>
       <element name="attachmentName" type="string"/>
       <element name="attachmentMimeType" type="string"/>
       <element name="attachmentURI" type="string"/>
     </sequence>
   </complexType>
```

```
</element>
```

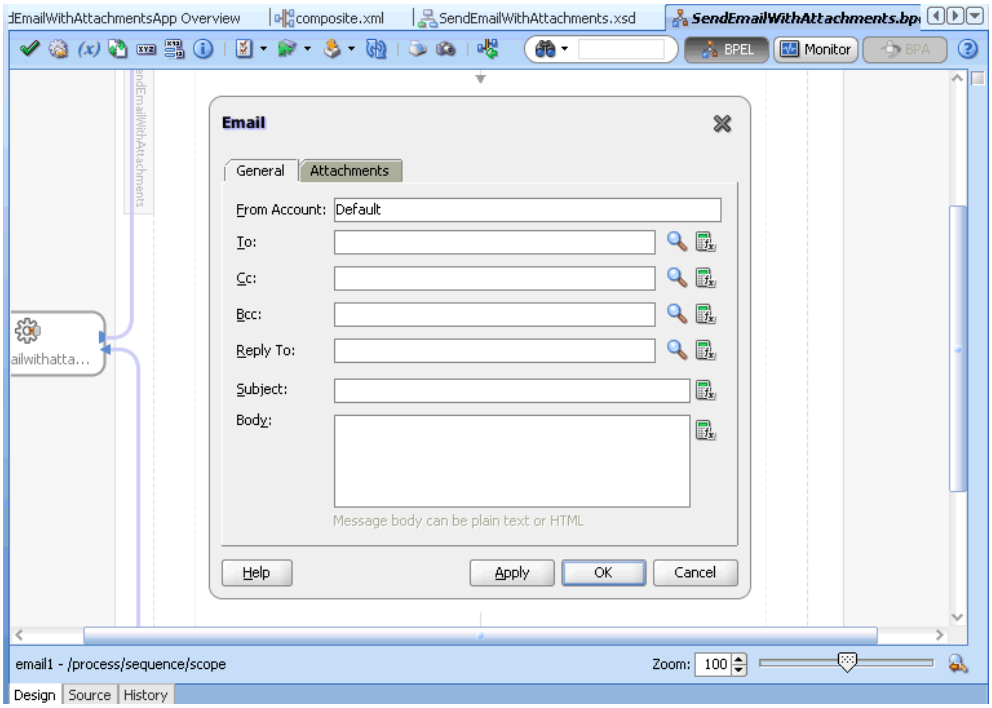
This xsd element defines the input for the BPEL process.

Figure J–26 Editing Email



12. Save the project.
13. Select the **SendEmailWithAttachments.bpel** editor screen.
14. Drag and drop an **Email** activity from **Oracle Extensions** located in the Component Palette between the **receiveInput** and **callbackClient** activities (Figure J–26).
15. In the Edit Email window, leave the From account as **Default**.

Figure J–27 Edit Email Window

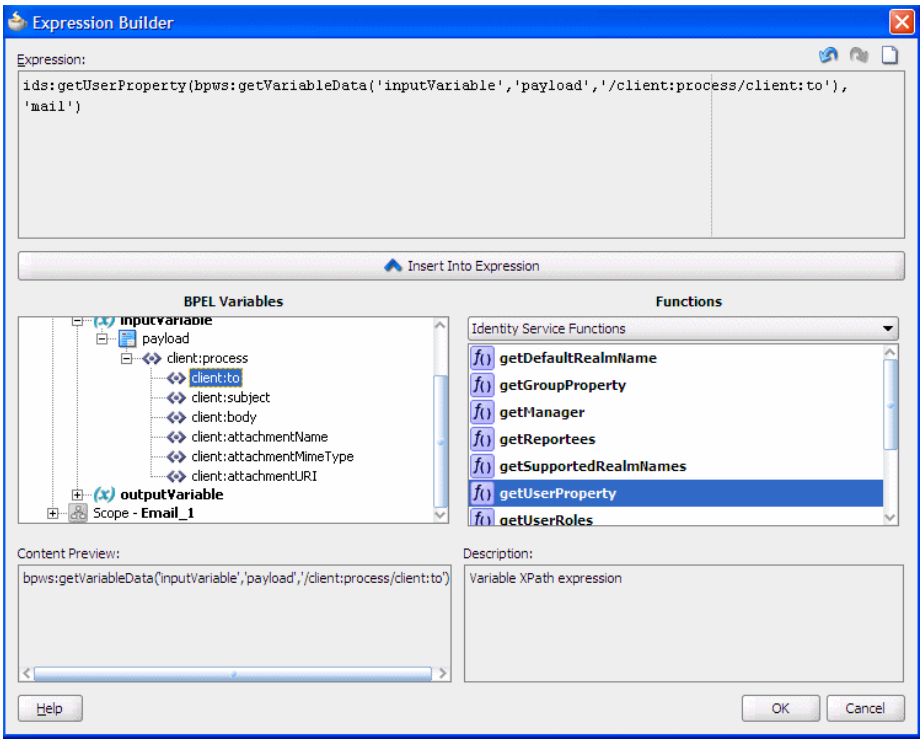


16. To create the expression for **To**, select the Expression Builder (the second icon, Figure J–28) and perform the following steps:

- Select **Identity Service Functions** from the functions dropdown list.
- Select the **getUserProperty()** function and select **Insert into Expression**.
- Under **BPEL variables** select **Variables** > **Process** > **Variables** >**inputVariable** > **payload** > **client:process** > **client:to**.
- Click **Insert into Expression**.
- Type the string `mail` manually.
- Correct the parenthesis so they are matched.
- Click **OK**.

This expression (Figure J–28) takes the data from the web service and maps it to the business email of the local SOA user.

Figure J–28 Expression Builder for the To Path



The expression should appear as follows:

```
ids:getUserProperty(  bpws:getVariableData('inputVariable','payload',
'/client:process/client:to'),'mail')
```

17. For **Subject**, select the Expression builder. Select **getVariableData** from **BPEL Extension Functions** and click **Insert Into Expression**.
18. Under **BPEL variables** select **Variables** > **Process** > **Variables** >**inputVariable** > **payload** > **client:process** > **client:subject**.

    The expression should appear as follows:

    ```
    bpws:getVariableData( 'inputVariable', 'payload','/client:process/
    client:subject')
    ```

19. For **Body**, select the Expression Builder. Select **getVariableData** from **BPEL Extension Functions** and click **Insert Into Expression**.
20. Under **BPEL variables** select **Variables** > **Process** > **Variables** >**inputVariable** > **payload** > **client:process** > **client:body**.

    The expression should appear as follows:

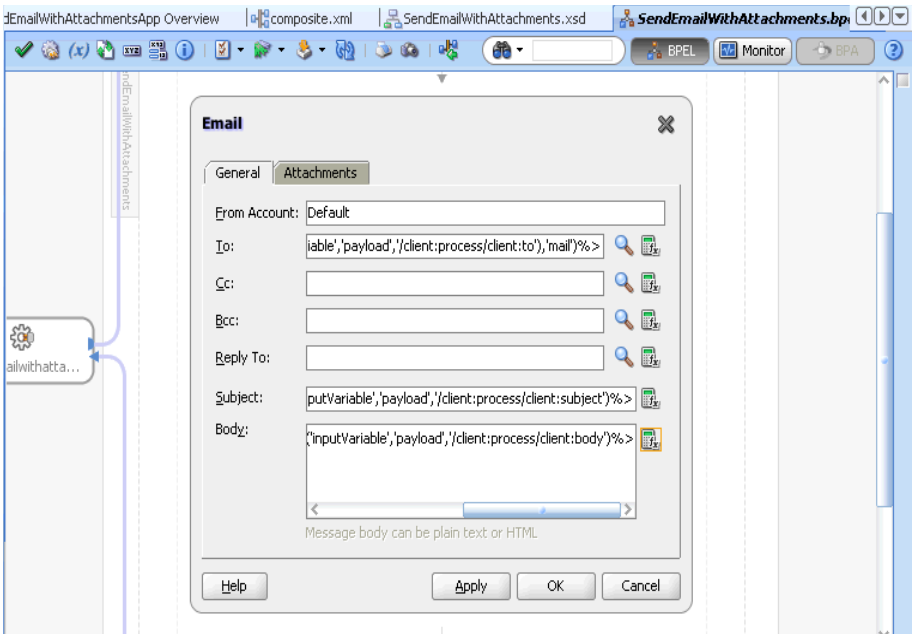
    ```
    bpws:getVariableData('inputVariable','payload','/client:process/client:body')
    ```

21. In the Edit Email dialog (Figure J–29), select the **Attachments** tab and use the add icon to add a specified number of attachments.

    When an email has multiple parts, the attachment count includes the body that is set with the Wizard above. The body specified by the Wizard above is set as the first body part.

    For example, to represent a multipart mail with one (1) attached file, specify `two` body parts. When there is one attachment, specify one body part.

Figure J–29 Edit Email Window



22. In the **Attachments** tab, set the attachments:

    Each attachment has three elements: name, MIME type, and value. All three elements must be set for each attachment.

    1. Click the **Attachments** tab.
    2. 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.)
    3. In the **Name** field, change the name or accept the default value of `Attachmentnumber`.
    4. In the **Mime Type** field, click the **Browse** icon to invoke the Expression Builder dialog for adding MIME type contents.
    5. When complete, click **OK** to return to the **Attachments** tab.
    6. In the **Value** field, click the **Browse** icon to invoke the Expression Builder dialog for adding the contents of the attachment.
    7. When complete, click **OK** to return to the Attachments tab.

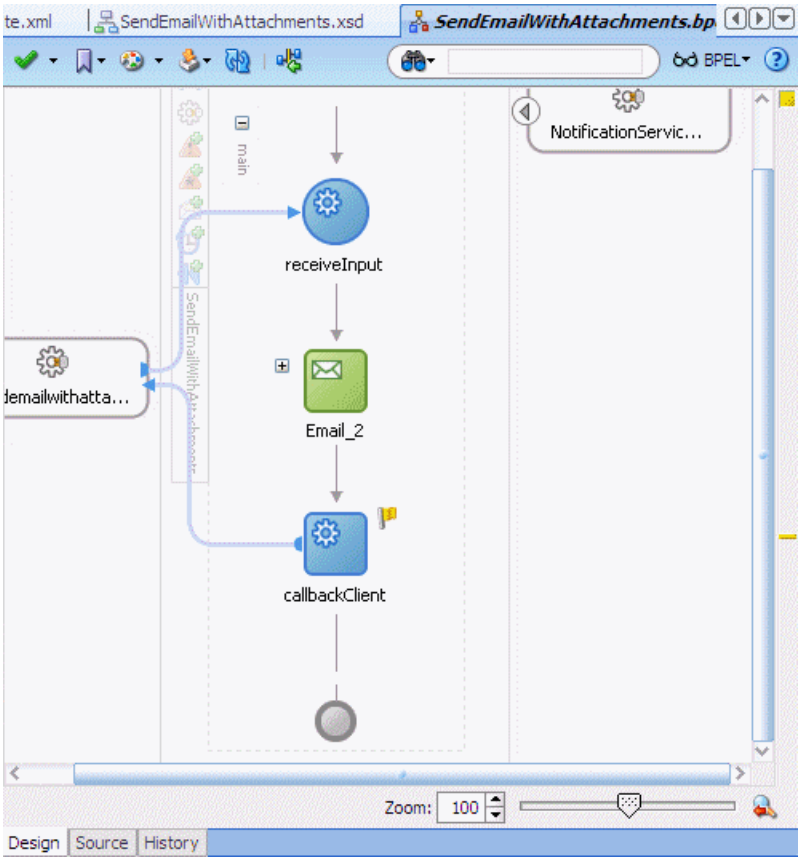
       The BPEL fragment with an assign activity with multiple copy rules is generated. One of the `copy` rules copies the attachment.

    8. Click **OK**.

    To view the default copy rules that were generated, and create new copy rules to transform the data:

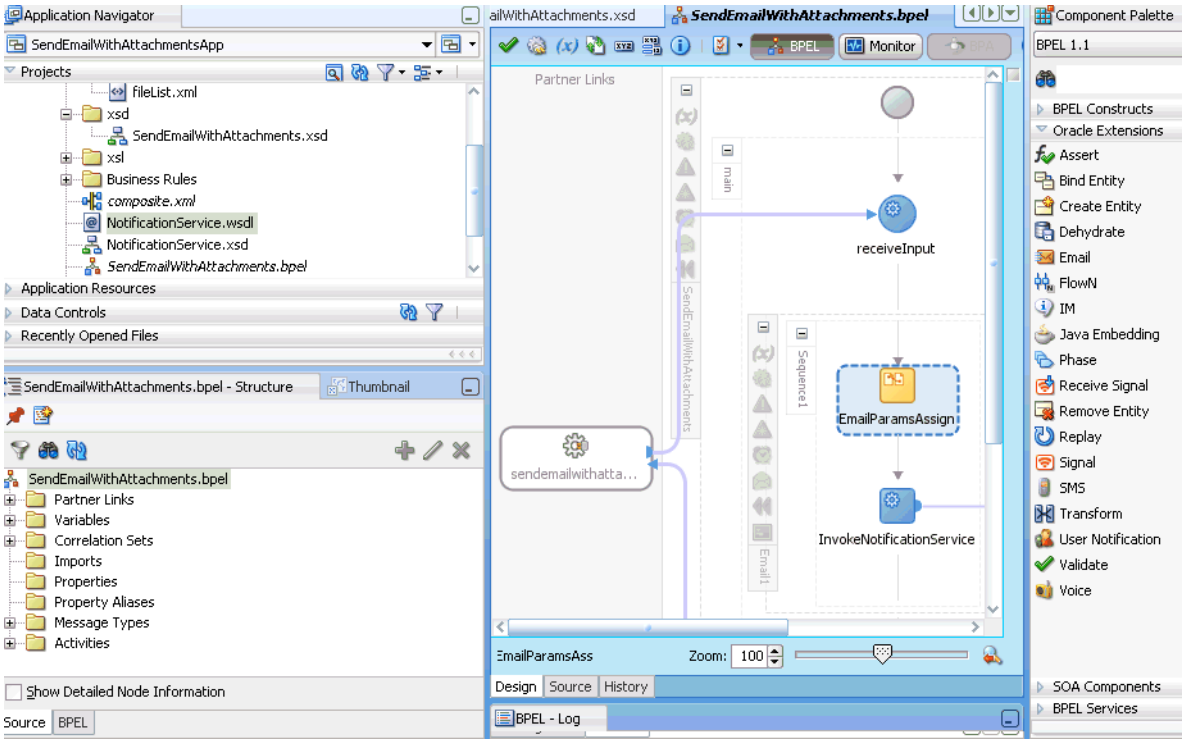
    1. Expand the Email node by selecting the plus sign icon (Figure J–30).

Figure J–30 Expanding the Email Node



2. Double-click the **EmailParamsAssign** node (Figure J–31).

Figure J–31 EmailParamsAssign Node



3. Insert a copy rule after the second attachment's **attachmentMimeType**. Under **Variables** select **Process** > **Variables** >**inputVariable** > **payload** > **client:process** > **client:attachmentMimeType**.

   Ensure that you select **Insert New Rule After**, and enter the **To XPath:** as shown below:

   ```
   From: /client:process/client:attachmentMimeType

   To: /EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/
   ns1:BodyPart[2]/ns1:MimeType
   ```

4. Click **Apply**.

5. Insert a copy rule after the second attachment's **attachmentName**. Under **Variables** select **Process** > **Variables** >**inputVariable** > **payload** > **client:process** > **client:attachmentName**.

6. Enter the **To XPath:** as shown below:

   ```
   From: /client:process/client:attachmentName

   To: /EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart
   /ns1:BodyPart[2]/ns1:BodyPartName
   ```

7. Click **Apply**.

8. Insert a copy rule after the second attachment's **body**. Under **Variables** select **Process** > **Variables** >**inputVariable** > **payload** > **client:process** > **client:body**.

9. Edit the **To XPath:** as shown below:

   ```
   From: ora:readFile(bpws:getVariableData('inputVariable','payload','/client:
   process/client:attachmentURI'))
   ```

```
To:
/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/
ns1:BodyPart[2]/ns1:ContentBody
```

10. Click **Apply**.
11. Click **OK** in the Edit Assign dialog.
12. Save the project.

The Process Modeling procedure is complete. You can use the information in this procedure to add notifications with binary attachments to your SOA composite application.

J.2.5.1 Sending Text Content with base64 Encoding

To send text file attachments with non-ASCII characters (such as UTF-8 encoded), you must send the text content with base64 encoding. Perform the following additional steps:

1. Click the BPEL source editor and add the following to the appropriate Body Part in the multipart content (look for correct `<BodyPart>` tag within the `<MultiPart>`):

```
<ContentEncoding
     xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
```

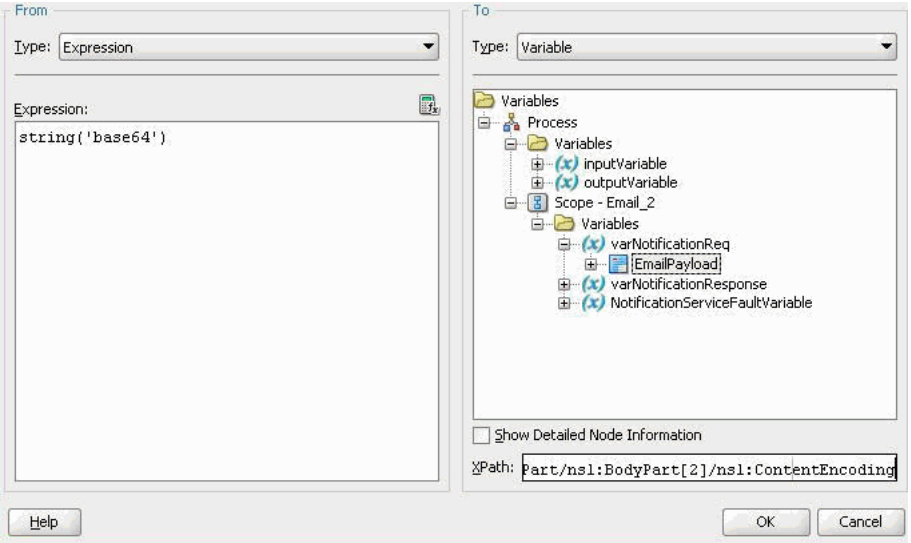
Example J–1 Adding the ContentEncoding tag to MultiPart

```
<copy xml:id="id33">
   <from xml:id="id31">
      <Content xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
         <MimeType xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
            multipart/mixed</MimeType>
         <ContentBody xmlns="http://xmlns.oracle.com/ias/pcbpel/
            NotificationService">
            <MultiPart xmlns="http://xmlns.oracle.com/ias/pcbpel/
            NotificationService">
             <BodyPart xmlns="http://xmlns.oracle.com/ias/pcbpel/
               NotificationService">
              <MimeType xmlns="http://xmlns.oracle.com/ias/pcbpel/
                NotificationService"/>
              <ContentBody xmlns="http://xmlns.oracle.com/ias/pcbpel/
                NotificationService"/>
              <BodyPartName xmlns="http://xmlns.oracle.com/ias/pcbpel/
                NotificationService"/>
             </BodyPart>
             <BodyPart xmlns="http://xmlns.oracle.com/ias/pcbpel/
                 NotificationService">
                <MimeType xmlns="http://xmlns.oracle.com/ias/pcbpel/
                   NotificationService"/>
                <ContentBody xmlns="http://xmlns.oracle.com/ias/pcbpel/
                   NotificationService"/>
             <BodyPartName xmlns="http://xmlns.oracle.com/ias/pcbpel/
                   NotificationService"/>
             <ContentEncoding xmlns="http://xmlns.oracle.com/ias/pcbpel/
                   NotificationService"/>
             </BodyPart>
           </MultiPart>
       </ContentBody>
    </Content
```

2. In the BPEL design editor expand the Email activity node and double-click the "EmailParamsAssign" node.
3. Select the "+" icon to create a copy rule which copies the "base64" string to our attachment part.
4. Edit the XPath as shown below:

```
From:
Type-Expresion: string('base64')

To:
Type-Variable:
/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:Con
tentEncoding
```

Figure J–32 Editing Copy Rules (2)



5. Click **OK**.
6. In the Assign window click **Apply** > **OK**.
7. Save the project.

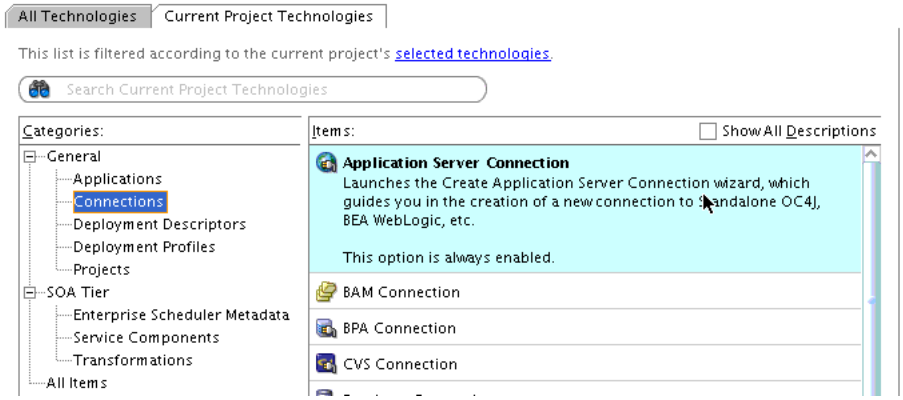
You can now deploy and run the application as described in Section J.2.3, "Running the Pre-Built Sample."

J.2.6 Creating a New Application Server Connection

Perform the following steps to create an Application Server Connection.

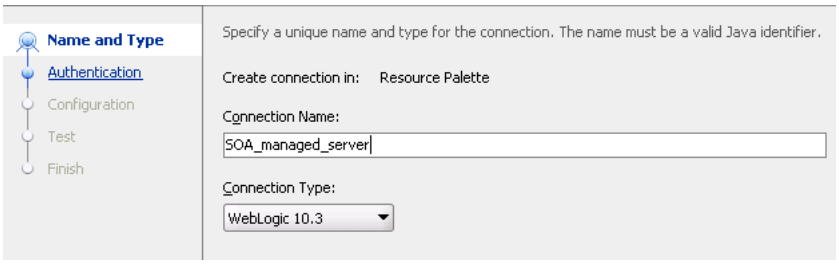
1. Create a new Application Server Connection by right-clicking the project and selecting **New**, **Connections**, and **Application Server Connection** (Figure J–33).

Figure J–33 New Application Server Connection



2. Name the connection `SOA_server` and click **Next** (Figure J–34).
3. Select **WebLogic 10.3** as the **Connection Type**.

Figure J–34 New Application Server Connection



4. Enter the authentication information. The typical value for username is `weblogic`.
5. On the Connection dialog, enter the hostname, port and SSL port for the SOA admin server, and enter the name of the domain for the Oracle WebLogic Server Domain.
6. Click **Next**.
7. In the Test dialog, click **Test Connection**.
8. Verify that the message `Success!` appears.

   The application server connection has been created.

K

Oracle SOA Suite Properties Road Map

This appendix provides an overview of Oracle SOA Suite design time and runtime configuration properties and provides references to documentation that describes how to configure these properties.

This appendix includes the following sections:

K.1 Oracle BPEL Process Manager Deployment Descriptor Properties

Deployment descriptors are BPEL process service component properties used at runtime by Oracle WebLogic Server, Oracle Enterprise Manager, or both. You set these properties during design time in the `composite.xml` file of the SOA composite application. The types of properties shown in Table K–1 can be set:

Table K–1 Properties for the configurations Deployment Descriptors

Property Name	Description
`completionPersistPolicy`	How to save instance data
`disableAsserts`	Whether to disable assertions in BPEL 1.1 projects
`globalTxMaxRetry`	The maximum number of retries for a remote fault
`globalTxRetryInterval`	The time interval in milliseconds between retries for a remote fault
`inMemoryOptimization`	In-memory optimization on the instances of a process
`keepGlobalVariables`	Whether the server can keep global variable values in the instance store when the instance completes
`oneWayDeliveryPolicy`	The persistence policy of the process in the delivery layer
`sensorActionLocation`	The location of the sensor action XML file
`sensorLocation`	The location of the sensor XML file
`transaction`	The transaction behavior of the BPEL instance for initiating calls

Table K–1 (Cont.) Properties for the configurations Deployment Descriptors

Property Name	Description
`idempotent`	An idempotent activity is an activity that can be retried (for example, an assign activity or an invoke activity). The instance is saved after a nonidempotent activity. This property is applicable to both durable and transient processes.
`nonBlockingInvoke`	Whether to spawn a separate thread to do invocations so that the invoke activity does not block the instance
`validateXML`	The enabling of message boundary validation

For more information about available deployment descriptor properties, see Section C.1.1, "How to Define Deployment Descriptor Properties" and Chapter 12, "Transaction and Fault Propagation Semantics in BPEL Processes."

K.2 Normalized Message Header Properties

Header manipulation and propagation are key business integration messaging requirements. You can set normalized message header properties during design time in the **Properties** tab of receive activities, invoke activities, OnMessage branches of pick and (for BPEL 1.1) scope activities, and reply activities. You can set properties for the following components:

- Oracle JCA adapters
- Oracle BPEL Process Manager
- Oracle Web Services Addressing
- Oracle B2B

For more information, see Appendix H, "Normalized Message Properties."

K.2.1 Oracle JCA Adapter Message Header Properties

Oracle JCA adapters expose the underlying back-end operation-specific properties as header elements and allow for manipulation of these elements within a business process.

For more information about available Oracle JCA adapter message header properties, see the following guides:

- Appendix A, "Oracle JCA Adapter Properties" of *Oracle Fusion Middleware User's Guide for Technology Adapters* for file, FTP, AQ, JMS, socket, database, and MQ Series properties
- *Oracle Fusion Middleware Adapter for Oracle Applications User's Guide* for Oracle Applications adapter properties

K.2.2 Oracle BPEL Process Manager and Oracle Web Services Addressing Message Header Properties

Oracle BPEL Process Manager and Oracle Web Services Addressing rely extensively on header support to solve customers' integration needs.

For more information about available Oracle BPEL Process Manager and Oracle Web Services Addressing message header properties, see Appendix H, "Normalized Message Properties."

K.2.3 Oracle B2B Message Header Properties

In B2B, you can manipulate headers with reserved key words.

For more information about available Oracle B2B message header properties, see Appendix C, "Back-End Applications Interface" of *Oracle Fusion Middleware User's Guide for Oracle B2B*.

K.3 SOA Composite Application Properties

While most updates you make to the `composite.xml` file are performed from within the dialogs of the SOA Composite Editor during design time, other properties must be added manually to this file from within **Source** view. Table K–2 lists these properties and provides references to documentation that describes how to configure these properties.

Table K–2 Oracle SOA Suite Properties

Property	Description	See...
`endpointURI`	Specifies multiple partner link endpoint locations. This capability is useful for failover purposes if the first endpoint is down.	Section 8.2.2.8, "Multiple Runtime Endpoint Locations"
`oracle.composite.faultPolicyFile`	Specifies the location of the fault policy file if it is different from the default location. This option is useful if a fault policy must be used by multiple SOA composite applications.	Section 11.4, "Using the Fault Management Framework"
`oracle.composite.faultBindingFile`	Specifies the location of the fault binding file if it is different from the default location. This option is useful if a fault policy must be used by multiple SOA composite applications.	Section 11.4, "Using the Fault Management Framework"
`passThroughHeader`	By default, SOAP headers are not passed through by Oracle Mediator. To pass SOAP headers, add this property to the corresponding Oracle Mediator routing service.	Section 19.2.2.9, "How to Assign Values" Section 19.2.2.11, "How to Access Headers for Filters and Assignments"
`rolesAllowed`	Specifies role names required to invoke SOA composite applications from any Java EE application.	Section 35.5, "Specifying Enterprise JavaBeans Roles"

`streamIncomingAttachments`
and
`streamOutgoingAttachments`
	Specify these properties to stream attachments with SOAP.	Section 42.1.1.2.1, "SOAP with Attachments"

`oracle.webservices.local.optimization`	Specifies to override a local optimization setting for a policy.	Section 42.1.1.2.1, "SOAP with Attachments" and *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*

Table K–2 (Cont.) Oracle SOA Suite Properties

Property	Description	See...
`one.way.returns.fault`	Controls how faults and one-way messages are handled for one-way interface SOAP calls.	Section 23.1, "Understanding a One-way Message Exchange Pattern"
`mtomThreshold`	Specifies the attachment size in bytes.	Section 42.1.1.3, "Adding MTOM Attachments to Web Services"

K.4 Fault Policy and Adapter Rejected Message Properties

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.

You can also enter additional properties in a fault policy framework file. Table K–3 lists these properties and provides references to documentation that describes how to configure these properties.

Table K–3 Oracle SOA Suite Fault Policy Properties

Property	Description	See...
`retryInterval`	Provides a delay between retries of an activity (in seconds).	Section 11.4.1.2, "Creating a Fault Policy File for Automated Fault Recovery"
`retryCount`	Retries an activity a specified number of times.	Section 11.4.1.2, "Creating a Fault Policy File for Automated Fault Recovery"
`org.quartz.scheduler.idleWaitTime`	Specifies a time in seconds for the scheduler to wait before retrying.	Section 21.1.1.2, "Actions"

You can also enter adapter rejected message properties in the fault policy framework file during design time.

For more information, see Section "Error Handling" of *Oracle Fusion Middleware User's Guide for Technology Adapters*.

K.5 Oracle B2B System Properties

You can set most B2B properties on the **Configuration** tab of the Oracle B2B interface. These settings override property settings performed at Oracle Enterprise Manager Fusion Middleware Control.

For more information about available Oracle B2B properties, see Chapter 15, "Configuring B2B System Parameters" of *Oracle Fusion Middleware User's Guide for Oracle B2B*.

K.6 Oracle Enterprise Manager Fusion Middleware Control Property Pages

You can configure properties for the following components during runtime in the property pages of Oracle Enterprise Manager Fusion Middleware Control:

- SOA Infrastructure

- Oracle BPEL Process Manager
- Human workflow notification and task service
- Oracle Mediator
- Cross references
- Oracle B2B
- Service and reference binding components (JCA adapters, web services, and Oracle Service Registry)

K.6.1 SOA Infrastructure Properties

You can configure properties for the SOA Infrastructure. These property settings can apply to all SOA composite applications running in the SOA Infrastructure. The following types of properties can be set:

- Audit level
- Composite instance state to capture
- Payload validation
- Universal Description, Discovery, and Integration (UDDI) registry
- Callback server and server URLs
- Instance and fault count metrics retrieval
- Java Naming and Directory Interface (JNDI) data source
- Web service binding properties

For more information about available SOA Infrastructure properties, see Chapter 3, "Configuring the SOA Infrastructure" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.6.2 Oracle BPEL Process Manager

You can configure BPEL process service engine properties. These properties are used by the BPEL process service engine during processing of BPEL service components. The following types of properties can be set:

- Audit trail level
- Audit trail and large document thresholds
- Dispatcher threads
- Payload schema validation
- BPEL monitor and sensor enabling

For more information about available Oracle BPEL Process Manager properties, see Chapter 9, "Configuring BPEL Process Service Components and Engines" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.6.3 Human Workflow Notification and Task Service

You can configure human workflow notification and task service properties. These properties are used by the human workflow service engine during processing of human workflow service components. The following types of properties can be set:

- The notification mode for messages

- The actionable addresses
- The actionable e-mail account name
- The workflow session time out and custom class path URL values
- The dynamic assignment and task escalation functions of the assignment service

For more information about available human workflow notification and task service properties, see Chapter 18, "Configuring Human Workflow Service Components and Engines" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.6.4 Oracle Mediator

You can configure Oracle Mediator properties. These properties are used by the Oracle Mediator service engine during processing of Oracle Mediator service components. The following types of properties can be set:

- Audit level and metrics level
- Parallel worker threads
- Parallel maximum rows retrieved
- Parallel locker thread sleep and error locker thread sleep
- Custom configuration parameters
- Container ID refresh time and container ID lease timeout
- Resequencer locker thread sleep, maximum groups locked, and worker threads

For more information about available Oracle Mediator properties, see Chapter 14, "Configuring Oracle Mediator Service Components and Engines" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.6.5 Cross References

You can configure cross references to dynamically map values for equivalent entities created in different applications.

For more information about available cross reference properties, see Chapter 15, "Managing Cross-References" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.6.6 Oracle B2B

You can enable Oracle B2B Dynamic Monitoring Service (DMS) metrics.

For more information about available Oracle B2B properties, see Chapter 30, "Configuring Oracle B2B" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.6.7 Service and Reference Binding Component Properties

You can configure the following service and reference binding component properties:

- Activation specification (for services), interaction specification (for references), and endpoint properties (such as time outs, thresholds, maximum intervals, and others) for the file, FTP, AQ, JMS, socket, database, and MQ Series adapters
- Web services properties such as enabling REST; enabling the WSDL, metadata exchange, and endpoint of the web service; and others

- Endpoint reference and service key properties for Oracle Service Registry integration

For more information about available service and reference binding component properties, see Chapter 33, "Configuring Service and Reference Binding Components" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.7 System MBean Browser Properties

The System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control enables you to modify advanced properties that do not display in the property pages described in Section K.6, "Oracle Enterprise Manager Fusion Middleware Control Property Pages." These advanced properties display beneath a link at the bottom of properties pages for the following components:

- SOA Infrastructure
- Oracle BPEL Process Manager
- Oracle Mediator
- Human workflow notification and task service
- Oracle Service Registry

> **Note:** In addition to advanced properties, the same properties that display for modifying in the property pages described in Section K.6, "Oracle Enterprise Manager Fusion Middleware Control Property Pages" also display for modifying in the System MBean Browser.

K.7.1 SOA Infrastructure Properties

Click the **More SOA Infra Advanced Configuration Properties** link at the bottom of the SOA Infrastructure Common Properties page to immediately display System MBean Browser properties for the SOA Infrastructure. Properties that display for modifying include the following:

- The maximum number of times an invocation exception can be retried
- The number of seconds between retries for an invocation exception
- The HTTP proxy authentication realm
- The HTTP proxy authentication type
- The HTTP proxy host
- The password for HTTP proxies that require authentication
- The HTTP proxy port number
- The user name for HTTP proxies that require authentication
- The HTTP protocol URL published as part of the SOAP address of a process in the WSDL file
- The HTTPS protocol URL published as part of the SOAP address of a process in the WSDL file
- The path to the Oracle SOA Suite keystore
- The UDDI endpoint cache life span

For more information about available SOA Infrastructure System MBean Browser properties, see Chapter 3, "Configuring the SOA Infrastructure" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.7.2 Oracle BPEL Process Manager Properties

Click the **More BPEL Configuration Properties link** at the bottom of the BPEL Service Engine Properties page to display System MBean Browser properties for the BPEL process. Properties that display for modifying include the following:

- The extra BPEL class path to include when compiling BPEL-generated Java sources
- The maximum number of times a failed expiration call (wait/onAlarm) is retried before failing
- The delay between expiration retries
- The size of the block of instance IDs to allocate from the dehydration store during each fetch
- The number of invoke messages stored in in-memory cache
- Whether one-way invocation messages are delivered
- The size of the most recently processed request list
- The maximum time a request and response operation takes before timing out

For more information about available Oracle BPEL Process Manager System MBean Browser properties, see Chapter 9, "Configuring BPEL Process Service Components and Engines" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.7.3 Oracle Mediator Properties

Click the **More Mediator Configuration Properties** link at the bottom of the Mediator Service Engine Properties page to display System MBean Browser properties for Oracle Mediator. Most of the System MBean Browser properties that display for Oracle Mediator can also be modified on the Mediator Service Engine Properties page.

For more information about available Oracle Mediator System MBean Browser properties, see Chapter 14, "Configuring Oracle Mediator Service Components and Engines" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.7.4 Human Workflow Notification and Task Service Properties

Click the **More Workflow Notification Configuration Properties** link at the bottom of the Workflow Notification Properties page or click the **More Workflow Taskservice Configuration Properties** link at the bottom of the Workflow Task Service Properties page to display System MBean Browser properties for human workflow. Properties that display for modifying include the following:

- The address at which to receive incoming instant messages (IMs)
- Whether to return custom notification service property names
- The return number of configured fax cover pages

For more information about available human workflow notification and task service System MBean Browser properties, see Chapter 18, "Configuring Human Workflow

Service Components and Engines" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

K.7.5 Oracle Service Registry WSDL URL Caching Configuration

You can increase the amount of time that the endpoint WSDL URL is available in cache for inquiry by the service key with the **UddiCacheLifetime** property.

For more information about the **UddiCacheLifetime** property, see Chapter 33, "Configuring Service and Reference Binding Components" of *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle BPM Suite*.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y